VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV MIKROELEKTRONIKY

DEPARTMENT OF MICROELECTRONICS

IMPLEMENTACE TVAROVÁNÍ ANTÉNNÍCH PŘÍJMOVÝCH SVAZKŮ RADARU V FPGA

RADAR RECEIVER BEAMFORMING IMPLEMENTATION IN FPGA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jakub Bárta**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Marek Bohrn, Ph.D.**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Mikroelektronika**
Ústav mikroelektroniky

Student: Bc. Jakub Bárta ***ID:*** 158102
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Implementace tvarování anténních příjmových svazků radaru v FPGA

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte a implementujte algoritmus příjmu dat jednotlivých datových streamů z anténních řad.
Navrhněte způsob synchronizace jednotlivých datových streamů.
Navrhněte a implementujte algoritmus tvarování (výpočtu vzorků) jednotlivých svazků.
Vyhodnoťte důsledky výpadku některého z datových toků (dat některé z řad antény).
Navrhněte a implementujte algoritmus odesílání výsledných dat.
Návrh proveďte konfigurovatelný za provozu (počet výstupních svazků, koeficienty rozložení amplitud, rozsah zpracovávaných vzorků).
Návrh proveďte pro FPGA Altera (Intel) v jazyce VHDL.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce

Termín zadání: 4.2.2019 ***Termín odevzdání:*** 21.5.2019

Vedoucí práce: Ing. Marek Bohrn, Ph.D.
Konzultant:

doc. Ing. Lukáš Fujcik, Ph.D.
předseda oborové rady

ABSTRAKT

Práce se v úvodu zabývá teorií radarů a jejich rozdělením. Dále představuje maticové antény, jejich parametry a možnosti práce s nimi. V hlavní části je proveden návrh systému pro výpočet tvarování příjímových svazků na FPGA Cyclone V a jeho ověření.

KLÍČOVÁ SLOVA

Radar, PSR, anténní řada,vyzařovací charakteristika, monopulz, tvarování svazku, FPGA, Altera, Cyclone V,

ABSTRACT

At the begining of this thesis radar theory and classification of radar systems is explained. Next part introduces antenna arrays with it's parameters and possibilities. Main part contains design of digital beamformer on FPGA Cyclone V and it's validation.

KEYWORDS

Radar, PSR, antena array,antena pattern, monopulse, beamforming, FPGA, Altera, Cyclone V,

BÁRTA, Jakub. *Implementace tvarování anténních příjmových svazků radaru v FPGA*. Brno, 2019, 59 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav mikroelektroniky. Vedoucí práce: Ing. Marek Bohrn, Ph.d.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Marku Bohrnovi, Ph.D. za trpělivost a konzultace k práci. Rovněž bych rád poděkoval firmě Retia a.s. především p. Davidu Párovi za zadání práce, poskytnutí vybavení a hodnotných informací nezbytných k vypracování této práce.

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této diplomové práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Vzhledem k ceně a složitosti konstrukce velkých směrových antén se v poslední době u aktivních radarů stále více využívá anténních soustav složených z mnoha malých antén uspořádaných do matice. To umožňuje zachování směrovosti a zisku při úspoře výrobních nákladů,monopulzní vyhodnocení cíle a v neposlední řadě také možnost elektrického tvarování vyzařovací charakteristiky antény, kterým lze do jisté míry nahradit mechanické ovládání. Tvarování anténního svazku se v současnosti provádí převážně číslicově. Oproti analogovému tvarování se přijatý signál zpracováním nezkresluje a netlumí. K digitálnímu tvarování je potřebný značný výpočetní výkon úměrný počtu elementů antény, zpracovávaných vzorků na periodu a množství vypočtených svazků. Pro výpočet jsou využity moduly signálových procesorů. Ty jsou však vzhledem k sériovému zpracování dat méně vhodné. Jejich kapacita se zvyšováním počtu paprsků přestává stačit. Nejjednosušším řešením je využití více výpočetních modulů pracujících paralelně. Tato možnost však není optimální z ekonomického a prostorového hlediska. Cílem této práce je návrh algoritmu, který umožní nahrazení současné soustavy signálových procesorů modulem s FPGA. Hradlové pole na rozdíl od procesoru umožňuje paralelní zpracování dat, které je pro tvarování svazku klíčové.

1 Radary

Název radar [1] vychází z anglické zkratky „radio detection and ranging“, což v překladu znamená rádiové zjišťování a zaměřování. Obecně jde o zařízení využívající elektromagnetického vlnění, jeho šíření a odrazů k zaměření a rozpoznání cíle. Z časového zpoždění vysokofrekvenčních pulzů přijatých anténou je stanovena vzdálenost cíle od radaru a ze změny frekence pulsu způsobené dopplerovým jevem rychlost cíle. U novějších přístrojů je za pomoci manipulace s vyzařovací charakteristikou za současného otáčení antény stanovena 3D souřadnice v prostoru. Radary se využívají například pro leteckou a lodní navigaci, vojenské účely, meteorologický průzkum nebo pro měření rychlosti vozidel.

Radary pracují většinou ve frekvenčním rozsahu od 3MHz do 3THz odpovídajícímu vlnovým délkám 100m-1mm. Použité frekvence jsou rozděleny do několika frekvenčních pásem, která jsou definována různými standardy.

Mezi nejvýznamější standardy patří:

- Standard mezinárodní telekomunikační unie (ITU)[31]
- EU-NATO-US standard
- IEEE 521-1984[32]

Rozdělění prekvenčních pásem a porovníní je v tabulce 1.

Tab. 1.1: Frekvenční pásma definovaná normami

f	IEEE	EU,NATO,US	ITU
0-3Hz	-	-	-
3-30Hz			ELF
30-300Hz			SLF
300Hz-3kHz			ULF
3-30kHz			VLF
30-300kHz			LF
300kHz-3MHz			MF
3-30MHz	HF		HF
30-250MHz	VHF		VHF
250-300MHz		B	
300-500MHz	UHF		UHF
500MHz-1GHz		C	
1-2GHz	L	D	
2-3GHz	S	E	
3-4GHz		F	SHF
4-6GHz	C	G	
6-8GHz		H	
8-10GHz	X	I	
10-12GHz		J	
12-18GHz	K_U		
18-20GHz	K		
20-27GHz		K	
27-30GHz	K_A		
30-40GHz			EHF
40-60GHz	V	L	
60-75GHz		M	
75-100GHz	W		
100-110GHz		-	
110-300GHz	mm		
300GHz-3THz	-		THF

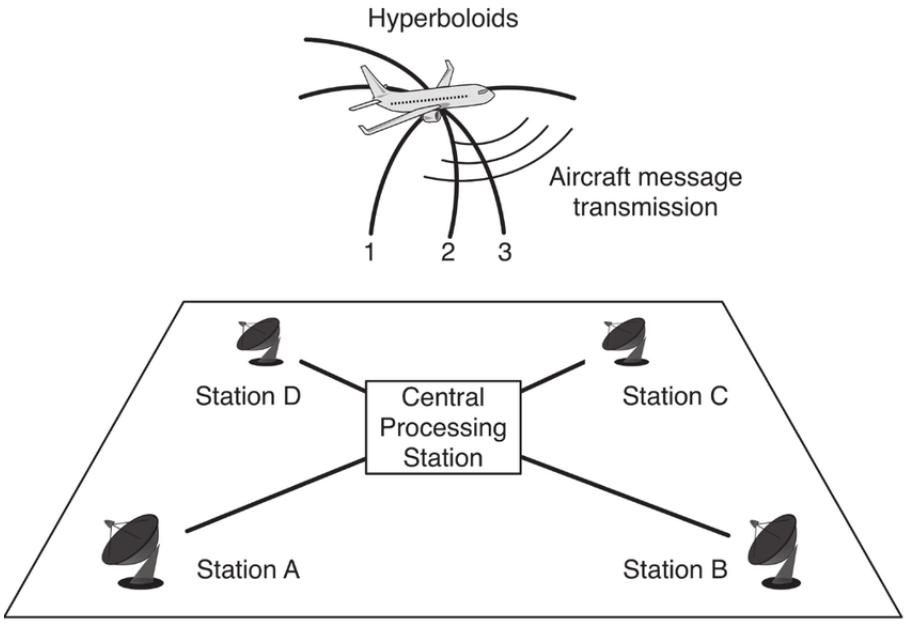
1.1 Základní rozdělení

Podle funkce se radary dělí do několika skupin. Podle toho, zda zařízení vysílá nebo pouze zachytává cizí signál se rozlišují radary aktivní a pasivní. Skupina aktivních

radarů se pak dále dělí na primární, přímající vlastní signál odražený od cíle, a sekundární, které přijímají odpověď generovanou cílem.

1.1.1 Pasivní radary

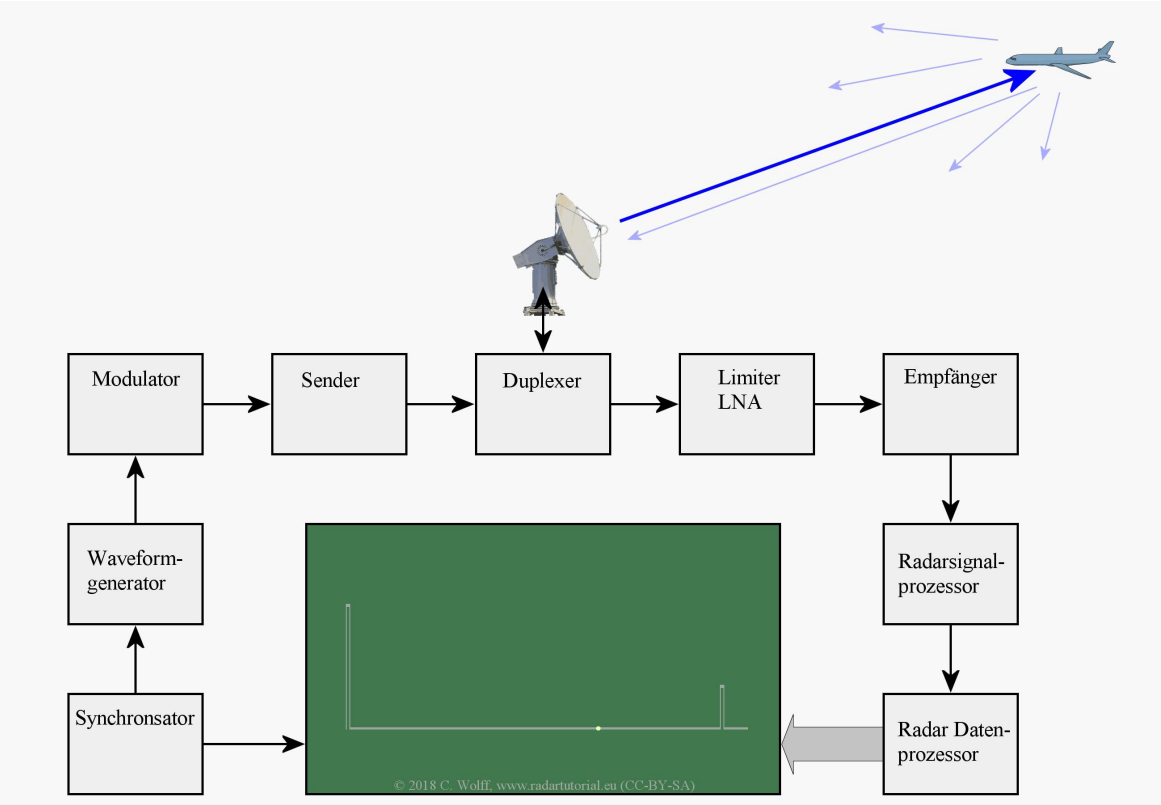
Pasivní radary[25] nevysílají žádný signál a k určení pozice cíle využívají pouze vysílání jiných zdrojů. Těmi může být například vlastní komunikace cíle, elektromagnetické záření motorů, odrazy signálu pozemních vysílačů nebo signál jiných (aktivních) radarů. Pro správnou funkci vyžaduje pasivní radar více přijímačů s různým umístěním ze kterých se za pomocí multilaterace zjistí poloha cíle. Multilaterace využívá k určení polohy hyperboly (resp. hyperboloidy), které se vyznačují stejným rozdílem vzdálenosti od bodů ohniska. Ohnisky jsou v případě pasivního radaru přijímače a rozdíl vzdálenosti je vypočten z časového posuvu přijatého signálu. Cíl se pak nachází v místě průniku hyperbol. Pro určení polohy v rovině je třeba použít 3 přijímače (2 hyperboly), pro práci v prostoru 4(3 hyperboly). Další zvyšování počtu přijímačů již ke zpřesnění nevede. Přesnost je dána především chybou měření časového rozdílu příjmu a uspořádáním přijímačů v prostoru. Hlavními výhodami tohoto typu jsou malé rozměry, možnost transportu, špatná lokalizovatelnost a odolnost vůči rušení. Příkladem mohou být české pasivní radiolokátory Tamara a Věra vyvinuté firmou Tesla Pardubice (dnes Era a.s.)



Obr. 1.1: Zaměření cíle pomocí multilaterace [24]

1.1.2 Primární radary

Primární radar [3]je určen k detekci nespolupracujících cílů. Princip spočívá v přijímání odrazů vyslaného signálu. Z časového zpoždění přijatého pulzu je poté stanovena vzdálenost cíle od antény, rychlost pomocí změny frekvence pulzu (dopplerův jev) a poloha (azimut a elevace) z natočení antény. Ta mívá většinou dobrou přesnost pouze v horizontálním směru. Elevace se tak zjišťuje z více než jednoho oběhu s různým vertikálním úhlem antény. Tento nedostatek je často kompenzován využitím více antén pomocí monopulzního vyhodnocení nebo anténní soustavou a tvarováním svazku. Dále je možné zjistit například velikost cíle z amplitudy přijatého signálu, nebo jeho tvar. Blokové schéma je na obrázku 1.2



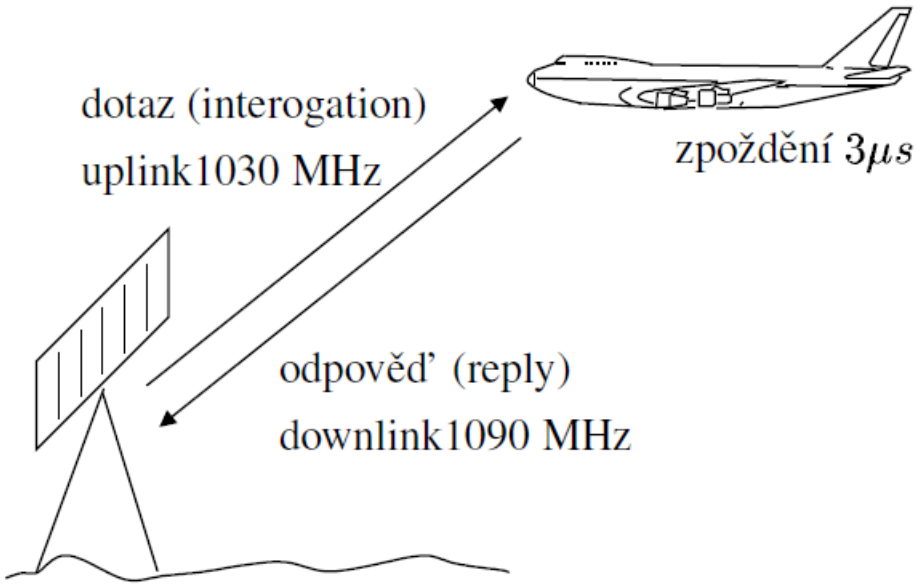
Obr. 1.2: Blokové schéma primárního radaru[2]

V první části cyklu je vygenerován signál, který je následně vyslán přes cirkulátor do antény. Cirkulátor je je zařízení sloužící k oddělování vysílaného a přijímaného signálu. Prakticky je možné představit si ho jako mechanický směrovač. Anténou je následně vyzářen pulz o výkonu v řádech kW. Po vyslání se zařízení přepne do přijímacího módu a zachytí odrazy signálu. Ten je dále zesílen a přesměrován do přijímače, ve kterém je většinou navzorkován a převeden do číslicové podoby. Ojediněle je možné setkat se i s analogovým zpracováním signálu, které je však na ústupu

vzhledem ke slábnutí a zkreslování signálu jeho zpracováním. Digitalizovaný signál je nejprve postoupen signálovému zpracování. Dále je společně se synchronizačními daty a informací o natočení antény předán systému pro detekci souřadnic cíle. Po zpracování je výsledek vykreslen na displeji přístroje.

1.1.3 Sekundární radar

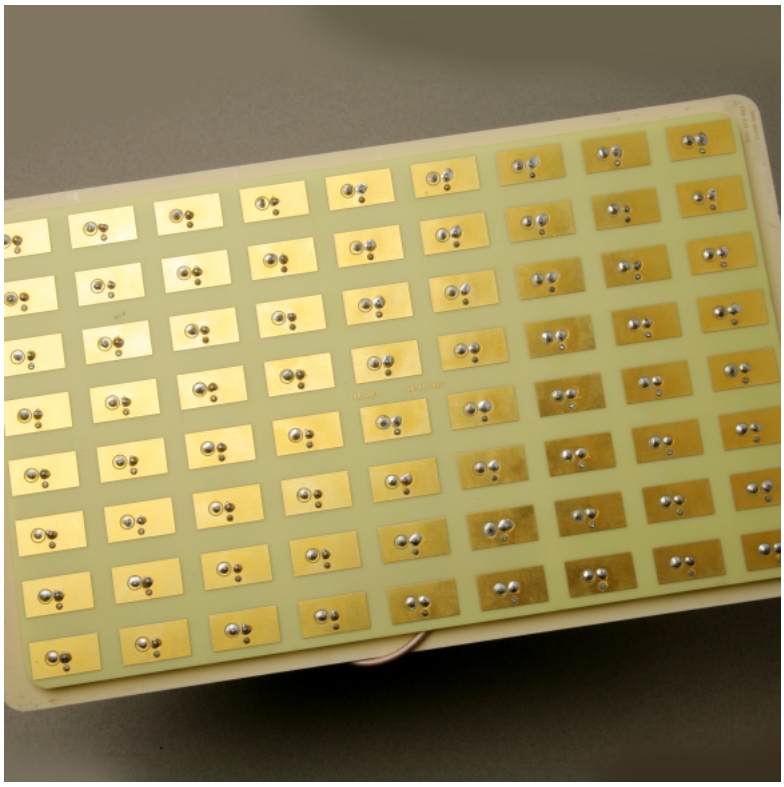
Sekundární radar [3] vznikl v době 2.světové války s hlavním cílem rozlišit přátelské a nepřátelské cíle. Na rozdíl od primárního radaru není schopen pracovat samostatně a pro svou funkci vyžaduje aby sledovaný cíl byl vybaven zpětným vysílačem (transpondérem). To umožňuje přenos více informací o cíli než jen poloha a rychlost. Toho se využívá v civilní letecké dopravě, kde se přenáší data z palubních přístrojů letadel obsluze letiště. Transpondér přijímá dotaz sekundárního radaru na frekvenci typicky 1030MHz a odpovídá na frekvenci 1090MHz aby nedocházelo k rušení příjmem odrazů, které jsou pro tento typ radaru nežádoucí. Využití rozdílné frekvence také umožňuje při použití dvou antén pracovat v kontinuálním režimu namísto pulzního. Pasivní radary pracují v různých módech, které se liší obsahem odpovědi.



Obr. 1.3: Princip sekundárního radaru[3]

2 Anténní řady

Pojmem anténní řada [4]se rozumí pole složené z obecně N oddělených antén vysílajících nebo přijímajících stejný signál. Využití soustavy umožňuje dosažení vyššího celkového zisku než u samostatné antény a lepší směrovosti. Dalšími výhodami je možnost elektronického řízení vyzařovací charakteristiky soustavy.



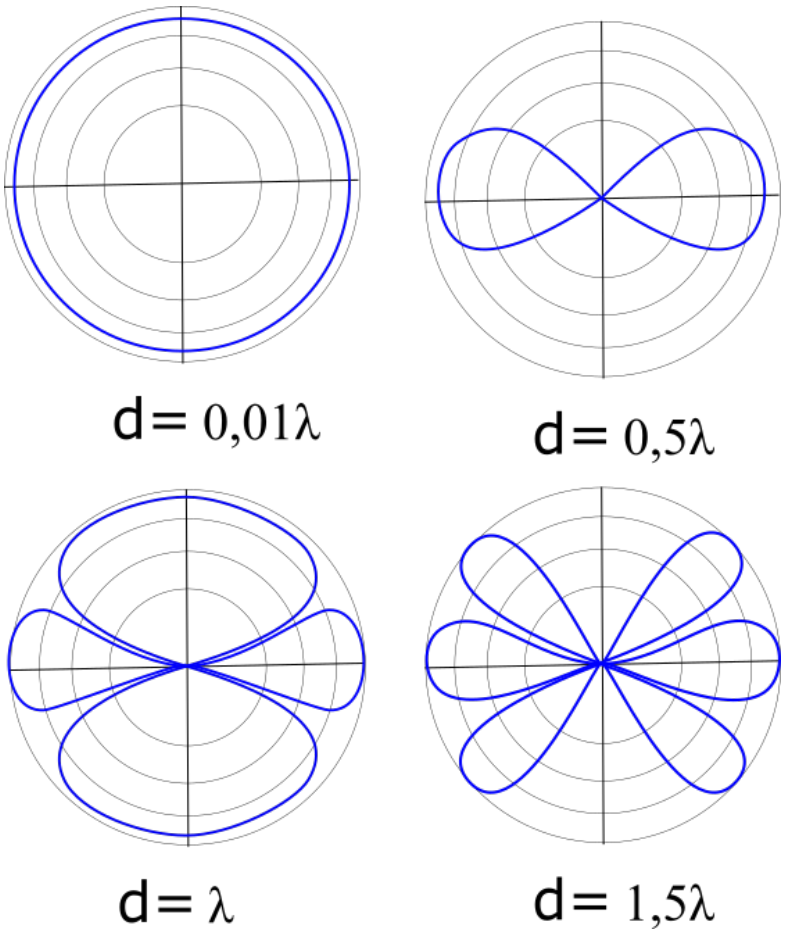
Obr. 2.1: Maticová anténa[4]

2.1 Vliv parametrů soustavy na směrovou charakteristiku

Vyzařovací charakteristika maticové antény závisí částečně na vlastnostech jednotlivých prvků, ale hlavně na jejich počtu a rozmístění v prostoru. V případě rozmístění prvků na přímce hovoříme o lineární řadě. V případě rozložení v rovině pak o plošné řadě.[16] Prvky je též možné rozložit po zakřivené ploše. Pak se hovoří o konformních řadách. U jednotlivých prvků předpokládáme obecně stejnou směrovou charakteristiku. Prvky můžou být tvořeny dipóly, trychýřovými anténami, ale bez problému i dalšími řadami. Příkladem může být Yagi anténa. Při návrhu řady je nutné nejprve

stanovit požadované pokrytí prostoru, pracovní frekvenci a zisk antény. Od toho se odvíjí plocha a počet prvků.

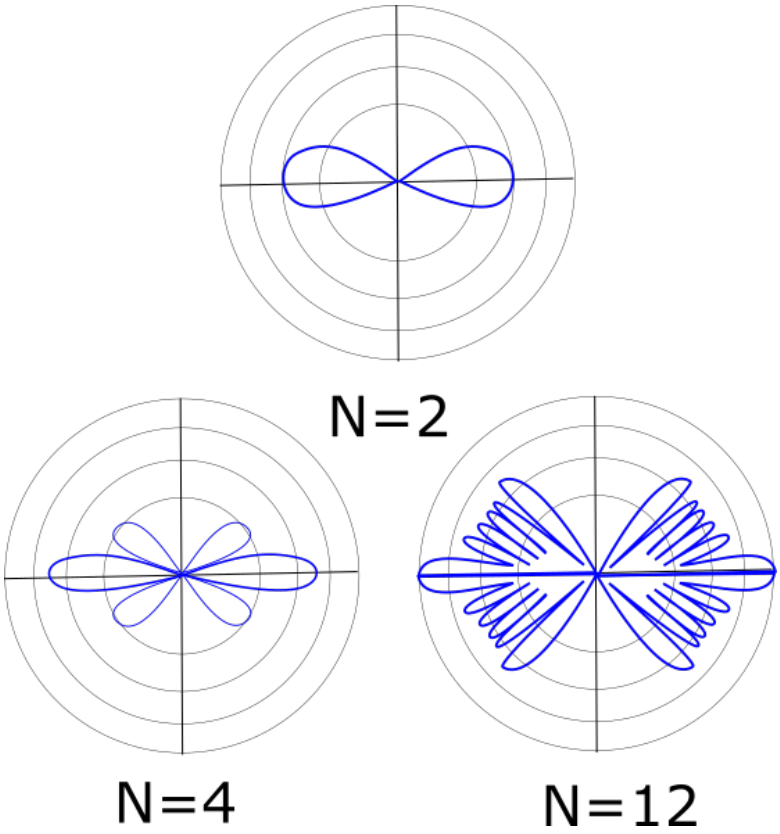
Vzdálenost prvků je odvozena od frekvence (resp. vlnové délky) vysílaného/přijímaného signálu. Používaný rozsah vzdáleností prvků je od desetin až po jednotky vlnové délky. Menší vzdálenosti způsobí ztrátu směrových vlastností antény a její vyzařovací charakteristika bude stejná jako u jednoho prvku. Větší vzdálenost bude zužovat hlavní lalok, ale vytvoří významné postranní laloky jejichž počet bude se vzdáleností prvků narůstat.[5] Vliv vzdálenosti elementů je prezentován na obrázku.



Obr. 2.2: Vyzařovací charakteristiky v závislosti na vzdálenosti elementů

V praxi se volí vzdálenost nejčastěji mezi 0.5 a 0,8 λ. U spodní hranice je hlavní lalok široký. Postranní laloky nevznikají, pokud mezi buzením prvků není fázový posun. Při fázovém posunu buzení dochází ke vzniku difrakčních postranních laloků. U horní hranice se hlavní lalok zúží za cenu vziku postranních laloků.

Dalším faktorem ovlivňujícím vyzařovací charakteristiku je počet prvků soustavy. Se zvyšujícím se počtem stoupá celkový zisk antény a hlavního laloku, ale zároveň se přidávají postranní laloky s nižším ziskem.[5] Vliv počtu prvků na vyzařivací charakteristiku je demonstrován na obrázku 2.8. Počet prvků se v praxi volí co největší pro dané rozměry a cenu.

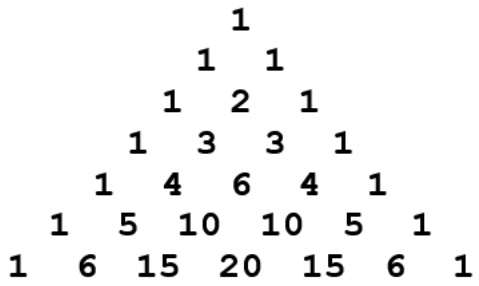


Obr. 2.3: Vyzařovací charakteristiky v závislosti na počtu elementů

2.2 Potlačení postranních laloků

Na celou řadu není vždy nutné aplikovat signál se stejnou amplitudou. Vhodným rozdělením amplitud na elementech řady mohou být zmenšeny nebo dokonce zcela odstaněny postranní laloky. Nevýhodou naopak může být rozšíření hlavního laloku (zhoršení směrovosti) a snížení celkového zisku antény (zmenšení odstupu signál-šum).[6], [7],[8],[9] Pro volbu rozložení amplitud v řadě je možné využít různé metody.

2.2.1 Binomické pole



Obr. 2.4: Pascalův trojúhelník

Prvním z možných rozdělění amplitud je využití koeficientů pascalova trojúhelníku (Obr:2.4).Tato metoda je nejvýhodnější z hlediska odstranění postranních laloků, neboť je odstaní úplně. Nevýhodou je omezená možnost aplikace vzhledem ke značnému rozptylu koeficientů, který znesnadňuje výpočty.

2.2.2 Dolph-Čebyševovo okno

Pro dosažení lepší směrovosti a menšího rozptylu hodnot se často využívá dolph-čebyševova funkce. Ta oproti Pascalovu trojúhelníku neodstraní postranní laloky kompletně, ale pouze omezí na stejnou úroveň. Rozdělení popisují rovnice 2.1 , 2.2, kde M je počet prvků antény a A požadované potlačení postranních laloků v dB.

$$w(\omega_k) = \frac{\cos\{M\cos^{-1}[\beta\cos(\frac{\pi k}{M})]\}}{\cosh[\frac{1}{M}\cosh^{-1}(\beta)]} \quad k = 0, 1 \ldots M-1 \tag{2.1}$$

$$\beta = \cosh\left(\frac{1}{M}\cosh^{-1}(10^{\frac{A}{-20}})\right) \tag{2.2}$$

Amlitudy funkce směrem od středu řady klesají, na krajích však při velkém počtu prvků dochází k prudkému nárůstu. Přesto, že se tento výpočet může zdát komplikovaný, je pomocí výpočetní techniky daleko lépe realizovatelný než pasalův trojúhelník. Hodnoty lze získat například v programu MATLAB funkcí *chebwin(L,r)[30]* obsaženou v System toolboxu , kde L je velikost pole a r Velikost potlačení postranních laloků.

2.2.3 Taylorova funkce

Taylorovo ($\overline{n}$)rozložení amplitudy vychází z Dolph-Čebyševova a bylo vytvořeno ve snaze získat "ideální rozložení". Je jakýmsi kompromisem mezi binomickým a Dolph-čebyševovým. Za cenu menší směrovosti řeší problém s nárůstem amplitudy u okrajů řady. Rozdílem je, že postranní laloky nejsou konstantní , ale postupně se zmenšují. Tato výhoda se uplatní především v případě dlouhé řady, kdy u Dolph-Čebyševa dochází ke zvětšení postranních laloků. Koeficienty jsou vypočteny podle rovnice 2.3. Volenými parametry funkce jsou počet prvků řady, řád Taylorova polynomu a velikost největších postranních laloků.

$$A(n) = 2 \cdot \sum_{1}^{M} F_m \cos\left(2\pi m \frac{k_n - \frac{1}{2}N + \frac{1}{2}}{N}\right) \tag{2.3}$$

$$F_m = \frac{-1^{(m+1)} \cdot a_m}{2b_m} \tag{2.4}$$

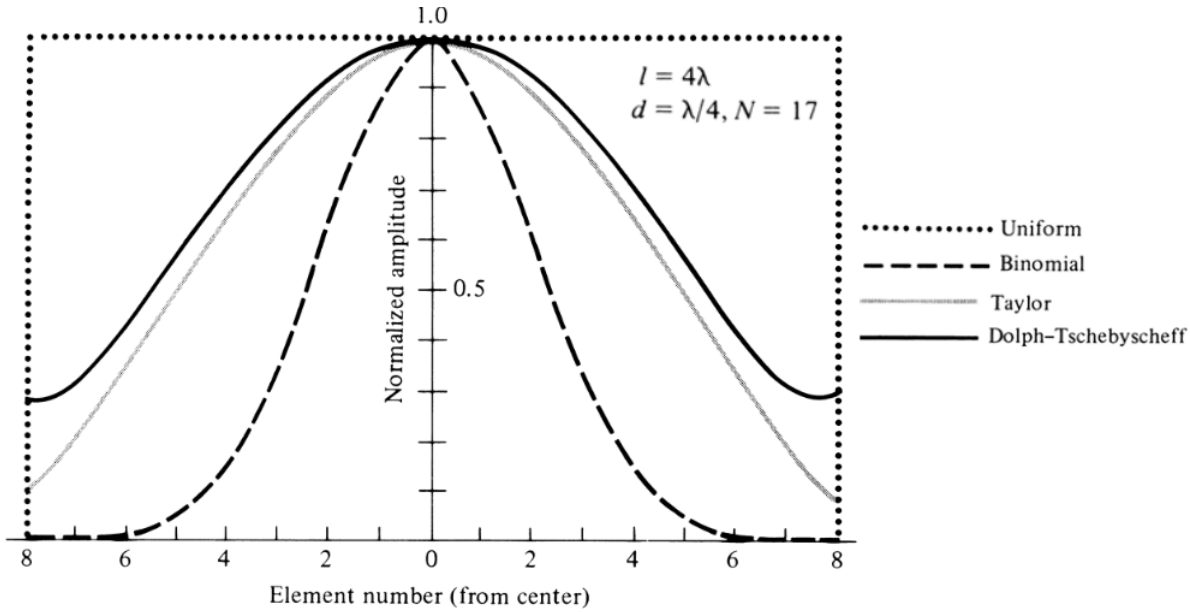
$$a_m = \prod_{x=1}^{M} \left(\frac{1 - \left(\frac{m^2}{\tau}\right)}{A^2 + \left(x - \frac{1}{2}\right)^2}\right); \tau = \frac{M^2}{A^2 + \left(M - \frac{1}{2}\right)^2}; A = \frac{\cosh\left(10^{-\frac{SLL}{20}}\right)}{\pi} \tag{2.5}$$

$$b_m = \prod_{y=1}^{M-1} \left(\frac{1 - m^2}{y^2}\right) \tag{2.6}$$

,kde:
M je řád Taylorova polynomu,
N je počet prvků řady,
$k \in \langle 0, N-1 \rangle$,
SLL je požadovaná velikost postranních laloků.

Pro praktické účely není stejně jako u Dolph-čebyševovy funkce nutné výpočet provádět ručně. Pro výpočet se běžně využívá funkce *taylorwin(L,nbar,sll)[29]*, kde L je velikost pole, nbar počet postranních laloků stejné velikosti, sll maximální velikost postranního laloku.

Porovnání výše zmíněných funkcí je viditelné na obrázku 2.5.



Obr. 2.5: Porovnání rozloženní amplitud [7]

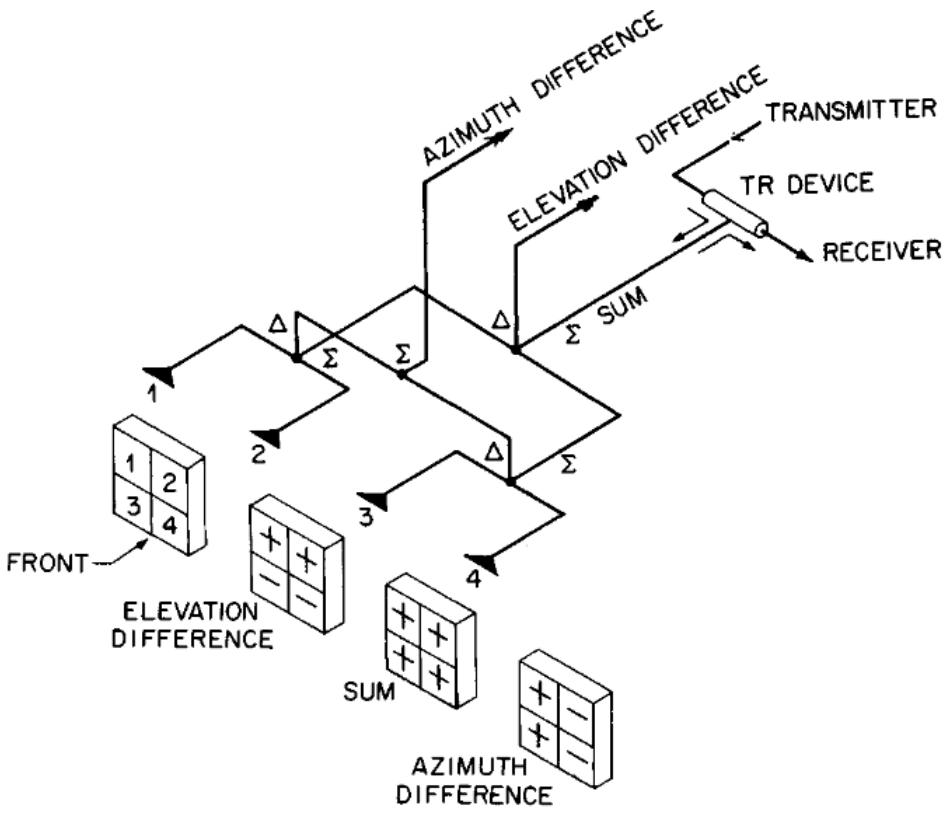
2.3 Monopulzní detekce cíle

Odražený radarový pulz dává informaci o vzdálenosti cíle od antény. Přesnou polohu v prostoru však nelze z jediného pulzu zjistit. Dříve se proto poloha zjišťovala pomocí tzv. kónického skenování. To spočívalo v postupném rotování paprsku radaru okolo cíle a korekci směru natočení antény.[2] Nevýhodou tohoto přístupu bylo především snadné zarušení. Během rotace totiž mohlo dojít ke změně podmínek (počasí, poloha cíle, vnější rušení). Výsledkem byla změna amplitudy přijatého signálu a následné špatné vyhodnocení cíle. Přesnost určení polohy totiž závisela hlavně na sledování růstu a poklesu síly přijatého signálu při pohybu paprsku. Zhruba od 60.let se proto u většiny radarů využívá monopulzní detekce cíle, která umožňuje vyhodnocení polohy cíle při vyslání a příjmu pouze jedné periody. Monopulzní vyhodnocení může být provedeno amplitudově nebo fázově.

2.3.1 Amplitudová syntéza

Amplitudová monopulzní detekce cíle vyžaduje minimálně 2 antény v každé ose. Pro plné určení polohy (azimut a elevace) jsou vyžadovány 4 antény uspořádané do čtverce. (Obr.2.6)

V případě anténní řady bude každý kvadrant brán jako zvláštní anténa. Výsledkem jsou částečně se překrývající paprsky. Směr je obvykle volen tak, aby se překrývaly v místě s poloviční intenzitou (-3dB). Pokud bude cíl přesně v ose an-



Obr. 2.6: Princip monopulzního radaru[26]

tény, bude na všechny části dopadat stejný signál. Když se však cíl vychýlí mimo osu, bude amplituda přijatého signálu různá. Na výstupu antény je obvod zajišťující sčítání a odčítání kvadrantů. Výstupem jsou součtový a 2 rozdílové signály.[26] Rozdílový signál pro elevaci je získán odečtením vertikálních polovin řady.

$$\Delta El = (S1 + S2) - (S3 + S4) \tag{2.7}$$

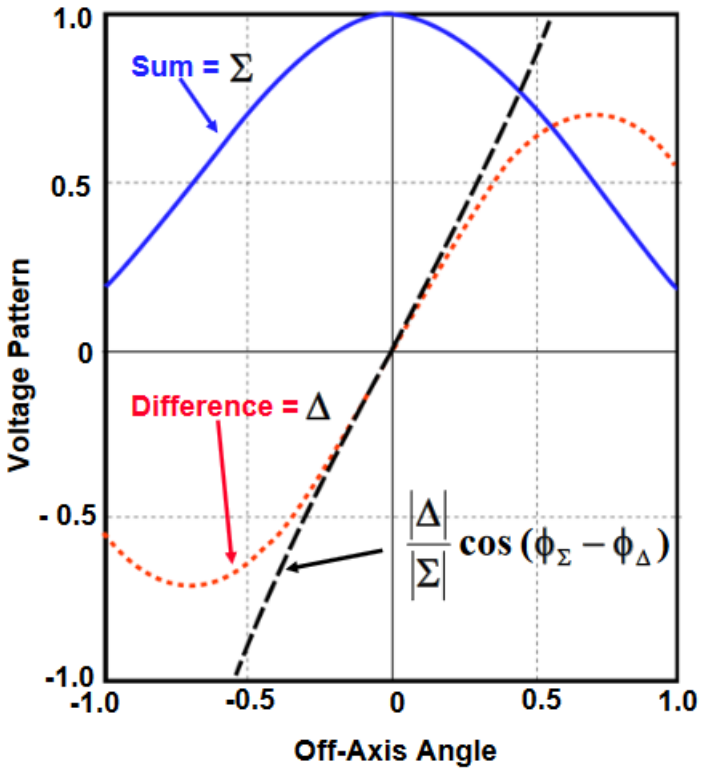
Rozdílový signál pro azimut je získán odečtšním horizontálních polovin řady.

$$\Delta Az = (S1 + S3) - (S2 + S4) \tag{2.8}$$

Součtový signál je součtem všech kvadrantů a slouží jako referenční signál pro určení úhlové citlivosti (V/ °).

$$\Sigma = S1 + S3 + S2 + S4 \tag{2.9}$$

Průběhy součtového a rozdílového signálu jsou znázorněny na obrázku 2.7. Čím strmější bude poměr rozdílového a součtového signálu, tím přesnější bude odhad chyby úhlu.



Obr. 2.7: Monopulzní detekce cíle[20]

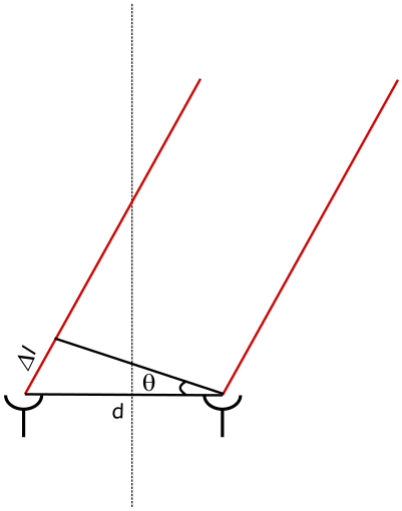
Zlepšení přesnosti amplitudové monopulzní syntézy je možné zajistit zvýšením strmosti rozdílového signálu. To se provádí aplikací váhové funkce jednotlivé prvky antény. Váhová funkce je volena tak, aby prvky vzálenější od středu, které mají na přesnost větší vliv, měly větší váhu.[11] Vhodnou velikostí váhové funkce se také kompenzuje pokles zisku na krajích antény způsobený amplitudovou modulací pro zlepšení chrakteristiky antény.(kapitola2.2)

2.3.2 Fázová syntéza

Pokud na anténní řadu dopadá signál pod určitým úhlem (obr.2.8), bude signál zachycený různými elementy řady fázově (časově) posunutý. Tento jev je způsoben rozdílem vzdálenosti Δl, kterou musí vlna překonat.

Časový posun Δl je dán vzdáleností prvků antény d a úhlem dopadu vlny θ 2.10.

$$\Delta l = d \cdot \sin\theta \tag{2.10}$$



Obr. 2.8: Dráhový rozdíl při dopadu signálu pod úhlem

Po dosazení zpoždění do rovnice vlny a úpravě bude pro n-tý prvek antény platit rovnice 2.11.

$$y_n(t) = x_n(t) \cdot \sin(2\pi ft + (n-1) \cdot \frac{2\pi}{\lambda} \cdot d \cdot sin\theta) \tag{2.11}$$

Kde y_n je výstup n-tého prvku řady v čase t, x je přijatý signál, f je nosná frekvence, λ příslušná vlnová délka a θ uhel dopadu signálu. Pro celou řadu je výstup dán rovnicí 2.12.

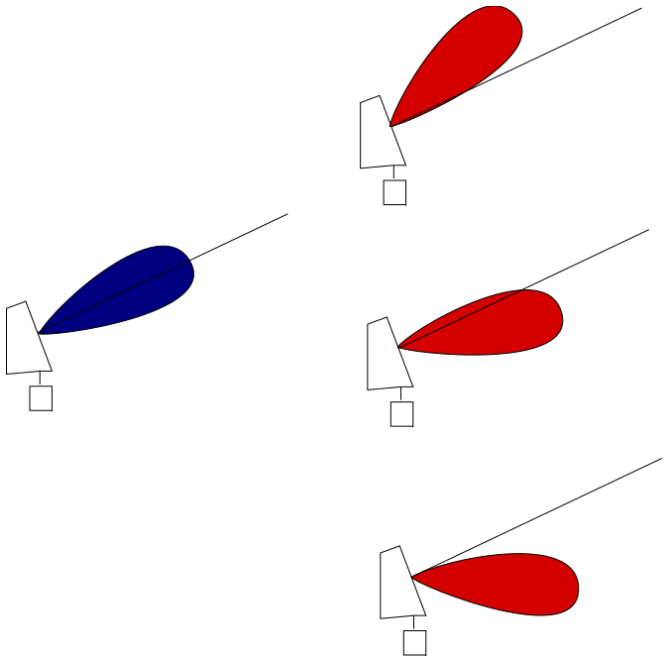
$$y(t) = x(t) \sum_{1}^{N} e^{j2\pi(n-1)\frac{d}{\lambda}\sin(\theta)} \tag{2.12}$$

Z takto získané funkce lze při známých parametrech antény a kmitočtu pulzu vypočítat polohu cíle.[5] Podmínkou je, aby vzdálenost prvků antény byla menší než $\lambda/2$ jinak budou prvky vzájemně ovlivněny. Přesnost vyhodnocení polohy u tohoto způsobu záleží především na parametru SNR (poměru signálu a šumu).

2.4 Beamforming

Fázovou funkci anténí řady lze využít nejen pro vyhodnocení polohy cíle, ale i pro řízení směru hlavního laloku antény. Dosazením libovolného úhlu do rovnice lze vypočítat signál přijatý z daného úhlu. Dochází tak vlastně k matematickému natočení

antény.(Obr:2.9) Stejný efekt bude mít i přidání časového posuvu k signálu přijatému jednotlivými elementy. Tento způsob elektronického vychylování je označován jako beamforming.[12] Tímto způsobem lze při digitálním zpracování z jednoho pulzu získat více různých příjmových svazků sloužících k upřesnění polohy cíle. Analogicky je možné ovlivnit časovým posuvem i úhel vysílácího svazku.



Obr. 2.9: Znázornění funkce tvarování svazku

Výpočet svazku pod určitým úhlem se provádí dosazením do rovnice. 2.13,

$$y(t) = x(t) \cdot \sum_{1}^{N} X(n) \cdot AF \tag{2.13}$$

,kde AF je část z rovnice 2.12 vyjádřená sumou. Ta bývá nazývána komplexní tvarovací váha a představuje přenos anténní řady. X(n) je amplitudová modulace pro zlepšení charakeristiky antény (viz kapitola.2.2)

Cílem navrženého beamformeru je realizace Výpočtu 3 rovnic pro součtový azimutární a elevační kanál:

$$B_{\Sigma}(k) = \sum_{1}^{N} \frac{a_n(k)}{s_{(n)DRU\Sigma S-A\Sigma}} \cdot S_{\Sigma}(n) \tag{2.14}$$

$$B_{\Delta A}(k) = \sum_{1}^{N} \frac{a_n(k)}{s_{(n)DRU\Delta S-A\Delta}} \cdot S_{\Delta}(n) \tag{2.15}$$

$$B_{\Delta E}(k) = \sum_{1}^{N} \frac{b_n(k)}{s_{(n)DRU\Sigma S-A\Sigma}} \cdot S_{\Sigma}(n) \tag{2.16}$$

,kde: B_{Σ} je signál součtového svazku,
$B_{\Delta A}$ je signál rozdílového azimutárního svazku,
$B_{\Delta E}$ je signál rozdílového elevačního svazku,
a_n je komplexní tvarovací váha n-té řady v azimutu,
b_n je komplexní tvarovací váha n-té řady v elevaci,
s_n jsou s-parametry n-té řady od fázového středu součtového/rozdílového kanálu k převodníku v DRU,
$S_{\Sigma}(n)$ je signál součtového kanálu n-té řady,
$S_{\Delta}(n)$ je signál rozdílového kanálu n-té řady,

Vztahy vychází z rovnice 2.12. Na rozdíl od odvození je v praktické aplikaci tvarovací váha navíc dělena s-parametry(viz kapitola 2.5) signálové cesty mezi výstupy antény a DTR modulem.

Komplexní váhy jsou vypočteny z rovnice2.17

$$a_n = X(n) \cdot e^{j2\pi \cdot (n:N-1) \cdot \frac{\Delta L_v}{\lambda} \cdot sin(\theta_k - \theta_0)} \tag{2.17}$$

,kde $X(n)$ je vektor modulace amplitudy
$n$je číslo anténní řady 1...N,
ΔL_v je rozteč prvků řady,
λ je vlnová délka signálu,
θ_k je úhel požadovaného paprsku,
θ_0 je odklon apertury

V rovnici pro elevaci se uplatňuje rozšířená komplexní váha pro zvýšení přesnosti monopulzního zpracování.(rovnice 2.18)

$$b_n = 2 \cdot a_n \cdot \left(\frac{2n - N - 1}{N - 1} \right) \tag{2.18}$$

2.5 s-parametry

V případě reálného radaru nezávisí signály pouze na venkovním prostředí, ale i na jeho cestě v rámci systému. Tu si lze zjednodušeně představit jako vícebran. Z teorie elektroniky je známé, že chování mnohabranu lze popsat skupinou parametrů získanou reálným měřením daného systému. K tomuto účelu většinou slouží Impedanční

Z-parametry, Admitanční Y-parametry nebo Hybridní H-parametry. Tyto parametry pracují s proudy a napětími a definují chování systému v případech kdy je jedna z bran naprázdno nebo nakrátko. U systémů pracujících na vyšších, typicky mikrovlných frekvencích toto není možné neboť zpravidla není možné dosáhnout stavu naprázdno nebo nakrátko aniž by docházelo k odrazům a interferenci vln. Dalším problémem je nestálý poměr napětí a proudu v různých místech systému. Z toho důvodu se pro popis vysokofrekvenčních zařízení využívají s-parametry [13], které namísto proudů a napětí řeší průchod normalizované zdrojové vlny systémem a její odrazy. Pro každou cestu mezi dvojicí bran, tvořených zpravidla koaxiálním vedením nebo vlnovody, existuje jeden parametr popisující přenos mezi těmito branami. Systém s N branami tak obsahuje N^2 parametrů. s-arametry jsou komplexní čísla, často v exponenciálním tvaru, představující amplitudu a fázi. signálu. Pro stanovené s-parametrů jsou definovány 2 standardizované vlny definované napětím, proudem a impedancí brány. Vlna působící na bránu i

$$a_i = \frac{V_i + Z_i I_i}{2\sqrt{\mid Re(Z_i)}} \tag{2.19}$$

a vlna odražená od i-té brány

$$a_i = \frac{V_i - Z_i^* I_i}{2\sqrt{\mid Re(Z_i)}} \tag{2.20}$$

,kde Z_i^* je komplexně sdružené číslo k impedanci. Pro stanovení s-parametrů pak platí

$$\begin{aligned} s_{11} &= \frac{b_1}{a_1}\Big|_{a_2=0} \\ s_{12} &= \frac{b_1}{a_2}\Big|_{a_1=0} \\ s_{21} &= \frac{b_2}{a_1}\Big|_{a_2=0} \\ s_{22} &= \frac{b_2}{a_2}\Big|_{a_1=0} \end{aligned} \tag{2.21}$$

Pro měření s-parametrů se využívá síťový analyzátor. Přístroj vybudí testovaný systém vysláním zdrojové vlny a na výstupu zaznamená průchozí vlnu. Síťové analyzátory mouhou měřit buď skalárně, kdy zaznamenají pouze amplitudy vln, nebo vektorově se záznamem komplexních parametrů. Pro správnou charakterizaci systému je nezbytné použití vektorového analyzátoru. [13]

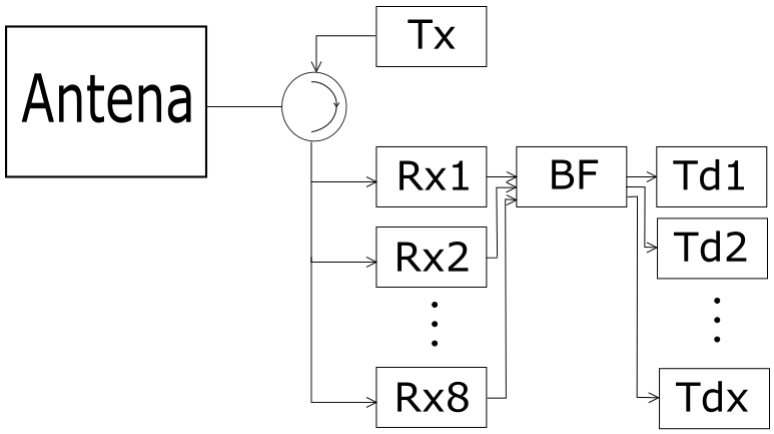
3 Návrh zařízení

3.1 Demonstrátor Re3D

Obr. 3.1: Demontrátor 3D radaru na střeše firmy retia a.s.

Prvotní nasazení tvarovače svazků je uvažováno pro demonstrátor 3D radaru [23], umístěný na střeše sídla firmy retia a.s. Zařízení je vybaveno plošnou anténou složenou ze 32 horizontálních řad a 8 vertikálních. Rozteč jednotlivých elementů je 73.1mm a odklon od vodorovné osy 15°. Radar využívá monopulzního vyhodnocování cíle v horizontálním směru a výstupem antény jsou tak součtový a rozdílový signál pro každou z vertikálních řad. Zařízení pracuje jako aktivní primární radar v pásmu S na frekvenci 3120MHz. Během jedné periody jsou vyslány 2 pulzy, SP(krátký pulz) pro blízké cíle a LP(dlouhý pulz) pro cíle vzdálené. Následně je anténa přepnuta do přijímacího módu a vyslaná perioda je přijata. Tyto signály jsou

vedeny do DTR modulu, který zajišťuje navzorkování a převod signálu 16bitovým A/D převodníkem. Převedený signál je odeslán do tvarovače,který ze širokopásmového signálu vytvoří několik paprsků jejichž hodnoty dále odesílá k vyhodnocení cíle. Blokové schéma je na obrázku 3.2.



Obr. 3.2: Blokové schéma demonstrátoru

Komunikace v rámci systému je řešena pomocí sběrnice Ethernet s využitím UDP protokolu. Data jsou odesílána ve stadndardizovaných paketech. Na výstupu DTR modulů, kde je uvažováno nasazení beamformeru je možné setkat se se třemi typy paketů:

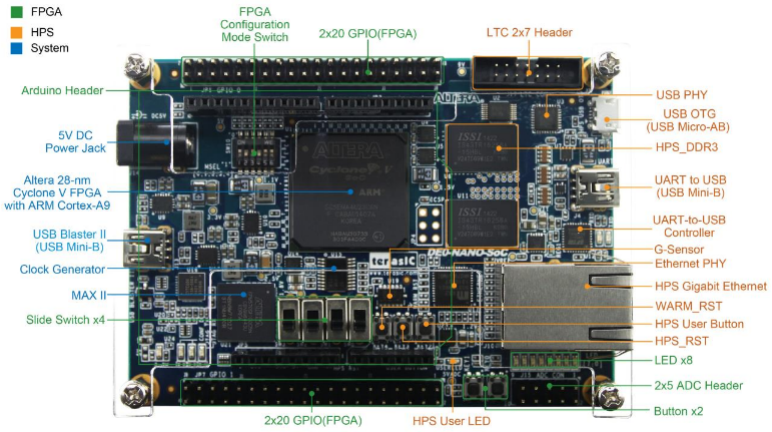
- Řídicí a stavové zprávy
- Přijatý signál
- Diagnostické zprávy

Každý z výše uvedených formátů má definovanou hlavičku obsahující informace o přenášených datech jako například číslo vertikální řady, typ kanálu (součtový/rozdílový) nebo počet datových bytů. Za kterou se nachází samotná data. Při zpracovávání je třeba dbát i na bytovou organizaci. Hlavičky mají totiž organizaci "big endian"zatímco data samotná "little endian".

V současné době se o tvarování stará několik modulů signálových procesorů. Ty jsou ale vzhledem k charakteru zařízení méně vhodné. Data přichází nárazově a je potřebné je co nejrychleji zpracovat a odeslat. Pro tuto činnost je daleko vhodnější využití FPGA, které umožňuje paralelní zpracování dat a stejný výpočetní výkon tak může být dosažen menším počtem výpočetních modulů. Ve finálním zařízení se uvažuje využití modulů s FPGA arria 10. Tyto moduly jsou však v době tvorby práce stále ve vývoji a návrh a ověření tak proběhne na vývojovém kitu de0-nano od firmy Terasic. V rámci této práce se uvažuje pouze dlouhý pulz a jeden úhel tvarování. K následnému rozšíření stačí vícenásobné využití návrhu.

3.2 Vývojová deska DE0-nano

Kit DE0-nano/Atlas Soc je univerzální vývojová platforma pro práci s obvody System on chip firmy altera. Deska je postavena okolo obvodu rodiny Cyclone V s důrazem na nízkou cenu a je určena především pro seznámení s obvody SoC FPGA. Náhled desky s popisem hlabních částí je na obrázku 3.3.



Obr. 3.3: Vývojový kit DE0-nano[17]

Z vybavení desky bude pro aplikaci beamformeru využit pouze obvod FPGA realizující výpočet a rozhranní ethernet pro komunikaci s okolím.

Jak je z obrázku viditelné tak fyzická stavba vývojového kitu nedovoluje připojit Ethernetové rozhraní přímo k FPGA, ale je možné ho využít pouze pomocí HPS (Hardwarového procesoru). Z tohoto důvodu bude část návrhu zařizující příjem a odesílání dat navržena v jazyce C pro tento procesor.

Pro usnadnění práce se sítí, která vyžaduje často i více vláken bude program běžet přes linux distribuci angstrom obsažené na paměťové kartě kitu. Při zpracování zároveň nebude možné dosažení požadované rychlosti a proto bude rychlost dočasně snížena na 10Mb/s. Toto nahrazení by nemělo mít vliv na realizaci výpočetní části, která je hlavním cílem této práce. Ve finální aplikaci pak bude HPS nahrazena pomocí IP jader 1G nebo 10G Ethernetu a pomocnými obvody.

3.2.1 FPGA Cyvlone V

Cyclone V je cenově nejdostupnější rodina obvodů firmy Intel. Mezi nabízenými obvody jsou nejen klasická FPGA, ale i SoC kombinující FPGA s hard IP ARM procesory. Jedním z nich je i 5CSEMA4U23C6N obsažený na vývojovém kitu. Ten kromě FPGA obsahuje rovněž dvoujádrový procesor s jádrem ARM Cortex A9. V tabulce je přehled dostupných zdrojů obvodu.[18]

Tab. 3.1: Dostupné zdroje obvodu 5CSEMA4U23C6N

ZDROJ	Dostupný počet
Počet logických elementů	40 000
Logických modulů	15 880
Registrů	60 376
DSP Blok s proměnnou přesností	84
Násobička 18x18	168
FPGA GPIO	145
HPS GPIO	181
ARM Cortex A9	2 jádra

Pro implementaci beamformeru je klíčový dostatečný počet hardwarových 16-bitových násobiček.Těch obvod nabízí 168 samostatně a dále pak 2 v každém DSP bloku, kterých je v ovodu obsaženo 84. Celkem je tedy možné využít 336 16-bitových násobiček.

3.3 Beamformer

3.3.1 Implementace rovnic beamformeru

Výpočet se skládá ze 3 hlavních částí: komplexní násobičky, sčítačky a balíku pro snadnou modifikaci generických parametrů modulu.

Data jednotlivých řad se násobí váhami spojenými s s-parametry jako komplexní čísla v algebraickém tvaru podle vzorce3.1.

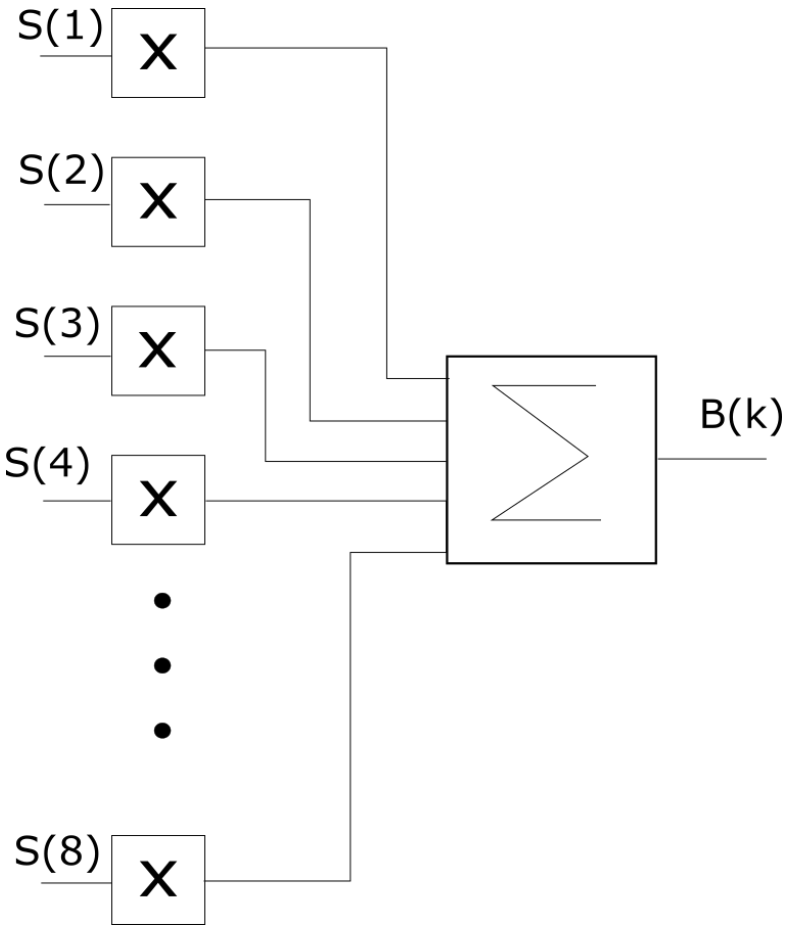
$$(a_1 + b_1 i) \cdot (a_2 + b_2 i) = (a_1 \cdot a_2 - b_1 \cdot b_2) + (a_1 \cdot b_2 + b_1 \cdot a_2)i \tag{3.1}$$

Násobení komplexních čísel obsahuje 4 operace násobení. Počet potřebných HW násobiček je $4 \cdot 8 = 32$ pro jeden kanál. Pro výpočet jednoho paprsku je nutné počítat 3 kanály (součtový, azimutární, elevační). Celý paprsek tedy vyžaduje $3 \cdot 32 = 96$ HW násobiček. Při využití daného obvodu je tak možné počítat maximálně tři paprsky paralelně.

Při součtu komplexních čísel v algebraickém tvaru se sčítají reálné a imaginární části zvlášť. Díky tomu je možné využít dvou jednoduchých 16 bitových sčítaček.

Výpočet rovnice jednoho kanálu je realizován zapojením N(8) násobiček a N(8) vstupé sčítačky. Blokové schéma je na obrázku3.4

Vstupem jsou data jednotlivých řad (z rovnic 2.152.162.14) a výstupem vypočtené svazky. Bloky násobení a sčítání jsou navíc doplněny o registry, aby byl



Obr. 3.4: Blokové schéma výpočtu

systém synchronní. Následně jsou paralelně zapojeny 3 tyto kanály (součtový, azimutární a elevační) pro vytvoření kompletního modulu beamformeru.

Balík BF_pkg

Jedním z požadavků na návrh je jeho modifikovatelnost. Ta je zajištěna parametrickým návrhem komponent. Pro snadnou změnu parametrů návrhu jako bitovou šířku vstupních dat, nebo počet řad antény jsou veškeré parametry přesunuty do balíku *beamformer.pkg*. Kromě globální definice paramatrů použití balíku umožňuje definici vektorových polí, které lze dále použít jako porty jednotlivých komponent. Modifikovatelnost návrhu je bohužel omezena použitím programu qsys(), jehož komplilátor pracuje pouze s minimem funkcí VHDL a neumožňuje využití parametrů. Návrh je proto zabalen do vrstvy *qsys_top*, která veškeré generické části nahrazuje pevnou šířkou. Změna parametrů beamformeru je tak ještě podmíněna změnou šířek v tomto souboru.

3.3.2 Komplexní váhy a s-parametry

Výpočet komplexních vah je proveden v prostředí matlab resp. klonu GNU Octave. Pro výpočet byla navržena funkce

[an,bn]=bfk (Lv,ap,f,N,mod,beam)

Do funkce se dosadí za *Lv* rozteč prvků anténí řady,
ap odklon apertury,
f pracovní frekvence,
N počet prvků řady,
mod vektor amplitudové modulace,
beam požadovaný úhel paprsku ve stupních.

Funkci je zároveň možné využít bez zadaných parametrů, kdy budou za nezadané parametry dosazeny hodnoty pro demonstrátor Re3D.(Kapitola 3.1) Jako výchozí amplitudová modulace je využito čebyševovo okno s potlačením postranních laloků 25dB.

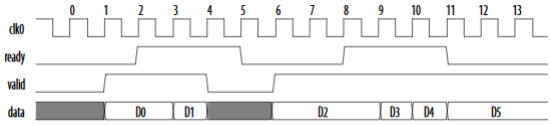
Výsledné komplexní matice jsou vstupem pro beamformer. Problém nastává s jejich charakterem. Běžné hodnoty komplexních vah se pohybují v rozmezí 2 a -2 obvykle s větším počtem desetinných míst. Pro interpretaci takovýchto hodnot se nejvíce hodí datový formát float, se kterým však bez dostatečné hardwarové podpory není možné pracovat. Další problém by nastal u nutnosti dělení vah s-parametry. Tato operace by byla ve formátu float prakticky neproveditelná a po převodu na integer by zabírala veliké množství zdrojů fpga. Hodnoty vah a s-parametrů jsou stálé a proto je jejich podíl možné počítat externě. Před použitím v systému je ještě aplikováno násobení 10^4 pro převod na 16bit Integer, který by měl zajistit dostačující přesnost a jednoduchou realizaci výpočtu. Tento přepočet je v rámci práce proveden v prostředí Octave s výstupem do textového souboru. Vypočtená data jsou uložena v paměti RAM připojené na vstup výpočetního modulu. Data do paměti jsou načtena z textového souboru po spuštění programu v HPS. Paměť ram je vhledem k její velikosti generována pomocí Python skriptu.

Pro správnost výsledků je před dalším zpracováním třeba převést výstup beamformeru na float s jednoduchou přesností (32bit) a dělění 10^4.

3.3.3 Avalon Stream

Data anténních řad pro výpočet musí být dodávána pokud možno co nejrychleji formou streamu aby s každým hodinovým taktem byla zpracována jedna hodnota. Stejně by tomu mělo být i v případě výstupu, který bude pouze o několik taktů

opožděn vlivem registrů vložených do výpočtu pro synchronní chod. Zároveň je nutné zajistit snadné propojení nejen s HPS použitému pro demonstraci, ale i s jeho náhradou ve formě IP jáder. Jako vhodné se jeví využití standardizovaného rozhraní Avalon Streaming®, kterým disponuje většina jader vyvinutých firmou Intel. Rozhraní Avalon streaming je vhodné pro jednosměrnou komunikaci s malým zpožděním. Zároveň je v případě potřeby využít i složitějsích funkcí pro řízení provozu na sběrnici. Komunikace má vždy 2 účastníky - vysílač(source) a přijímač(sink). Pokud má vysílač aktivní vstup *ready*, s hodinovým signálem odesílá data na sběrnici a signálem *valid* potvrzuje jejich platnost. Přijímač data zachytává a v případě neschopnosti dalšího příjmu zabrání shozením signálu *ready* dalšímu vysílání. Časový průběh je na obrázku3.5

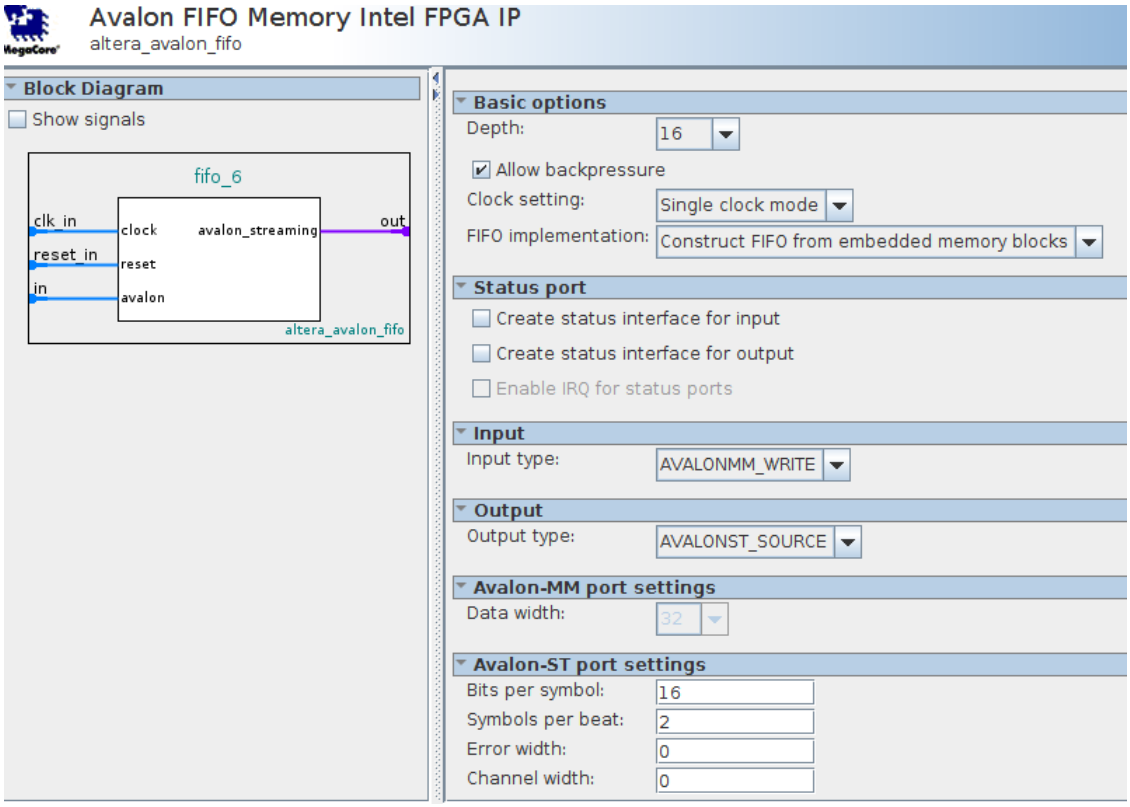


Obr. 3.5: Časový průbeh komunikace na sběrnici Avalon Stream[19]

Pro návrh to znamená rozšíření výpočetního modulu o signály *ready* a *valid*. Signál *valid* beamformeru je aktivní v případě, kdy jsou aktivní všechny *valid* signály na vstupu a signál *ready* na výstupu. Pro správnou funkci je signál *valid* zpožděn o počet registrů v signálové cestě aby jeho stav odpovídal platným vstupům. Signál *ready* je spojen se signálem *enable* vstupních registrů beamformeru a je aktivní v případě kdy jsou aktivní všechny vstupy *valid* (data na vstupu jsou platná).

3.3.4 Datová paměť

Jako rozhraní mezi HPS/IP core a Vypočetní modul je nutné využít paměť, do které je možné uložit data až do doby než dorazí pakety z DTR modulů všech řad. Datové pakety obsahují vždy celou sadu dat, která se zpracovává postupně a proto je vhodné využití paměti FIFO. Tuto paměť je možné vytvořit ručně pomocí VHDL kódu. Lepším řešením je však využití již hotových IP jader, které nabízí firma intel v rámci vývojového prostředí quartus. Tato jádra mají kromě ulehčení práce ještě tu výhodu, že disponují standardizovanými rozhraními intel Avalon®, která umožňují propojení mezi různými IP jádry. Využití takovéto paměti umožní pozdější nahrazení HPS IP jádrem bez nutnosti modifikace výpočetní části. Daší drobnou výhodou je také fakt, že intel ke svým jádrům dodává rovněž knihovny pro jejich využití v rámci HPS. Pro vytvoření paměti je ve vývojovém prostředí dostupný jednoduchý konfigurátor.(Obr.3.6) Hlavními parametry jsou bitová šířka



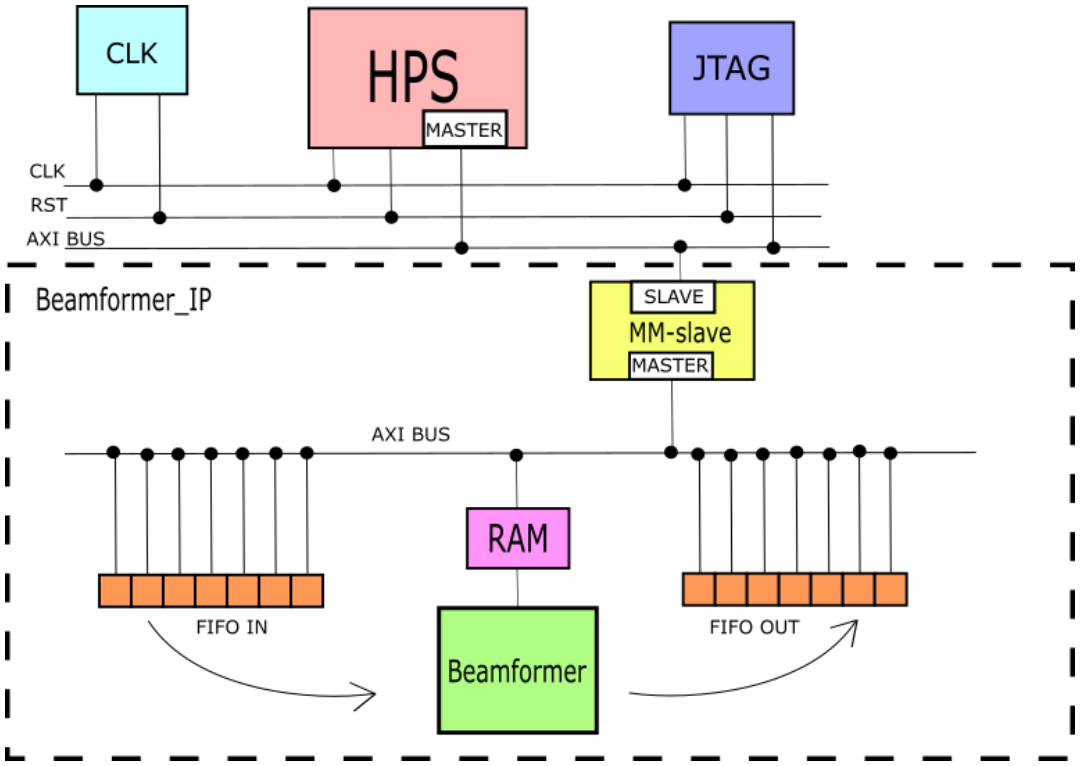
Obr. 3.6: Nstavení parametrů IP bloku paměti FIFO

slova, počet slov na takt, délka paměti a počet hodinových vstupů. Důležité je také povolení zpětného tlaku, které zajistí aby v případě přeplněné paměti se již další data neukládala. Volitelně je možné přidat rozhranní pro zjišťování stavu paměti, které může pomoci při ladění systému.

3.4 Top vrstva

Pro vytvoření top hierarchie projektu a propojení jednotlivých komponent návrhu slouží v prostředí intel quartus utilita platform designer(qsys). Pomocí grafického rozhraní je zde možné propojit jednotlivá ip jádra a zadat jejich parametry. Pro vložení do qsysu je nutné celý projekt překomplilovat v tomto prostředí a vytvořit z něj standardizované IP jádro. Nevýhodou je značné omezení podpory VHDL standardů což znemožňuje využití některých novějších funkcí. V případě beamformeru je kvůli omezení nutné zabalení komponenty do další vrstvy s pevně danou šířkou portů neboť qsys nepodporuje parametry v definici portu. Při změně bitových šířek nebo počtu řad bude nutné kromě změny parametrů balíku změnit i definici portů v top vrstvě pro qsys.

Po kompilaci následuje nastavení vstupních a výstupních portů v souladu se standardy Intel, aby bylo možné vzájemné propojení IP jader. Blokové schéma celého projektu je na obrázku 3.7. Obrázek přibližně odpovídá vzhledu schématu projektu v platform designeru.



Obr. 3.7: Blokové schéma projektu

Propojení jader s procesorem je realizováno pomocí AXI sběrnice typické pro procesory ARM-Cortex A9. Jádro beamformeru je navíc propojeno přes komponentu *memory mapped slave*. Díky tomu se pro procesor jeví vstupy beamformeru jako vlastní registry. Vstupy FPGA jsou tak snadno přístupné pro zápis i čtení programem běžícím v procesoru.

3.5 Zpracování udp paketů

UDP je jeden z protokolů Ethernetu. Na rozdíl od TCP nenavazuje spojení a vysílá data bez čekání na potvrzení příjmu. Výhodami tohoto způsobu je především jednoduchost komunikace a malé zatížení sítě. Nevýhody jsou možnost ztráty nebo poškození dat během přenosu.

Pro příjem je nejprve nutné otevřít socket na portu ze kterého je očekáván příjem. To se provádí funkcí z knihovny *sys/socket*. V případě více možných portů musí být pro každý port zvláštní socket. Samotný příjem probíhá příkazem *recvfrom* Příkaz po spuštění čeká až do příjmu paketu. Proto při přijmu z více zdrojů je nutné ho spouštět pro každý socket v jiném vlákně. Po přijetí jsou data uložena v datovém poli. Přijaté pakety mají předem definovaný tvar. Přibližný tvar paketu obsahujícího operační data je v tabulce 3.2.

Tab. 3.2: Struktura přijímaného paketu

Byte	Jméno	Dat.typ	Popis
1-4	Id	Uint32	Číslo odběhu
5	Row	Uint8	Číslo řady
6	Col	Uint8	Číslo sloupce
7	chan	Uint8	Kanál (rozdílový/součtový)
8	Pulse	Uint8	Pulz (dlouhý/krátký)
9-10	RSO	Uint16	Ofset dálkového segmentu
11-12	RSL	Uint16	Počet datových bytů (N)
N	QDt	Sint16	Imaginární data
N	InDt	Sint16	Reálná data

Hlavička se ukládá pro účely odeslání zatímco data signálu jsou převedena do vstupu FIFO paměti pro danou řadu. U dalších příchozích paketů je kontrolováno číslo odběhu. Příjem dat jiného odběhu způsobí odstranění veškerých dat z pamětí a nastavení tohoto odběhu jako aktuálního. Zabrání se tím míchání dat různých odběhů. K takovémuto vymazání dojte pravděpodobně pouze v případě ztráty dat některé řady. V jiném případě se nemůže stát, že by z některé řady přišel další odběh předčasně.

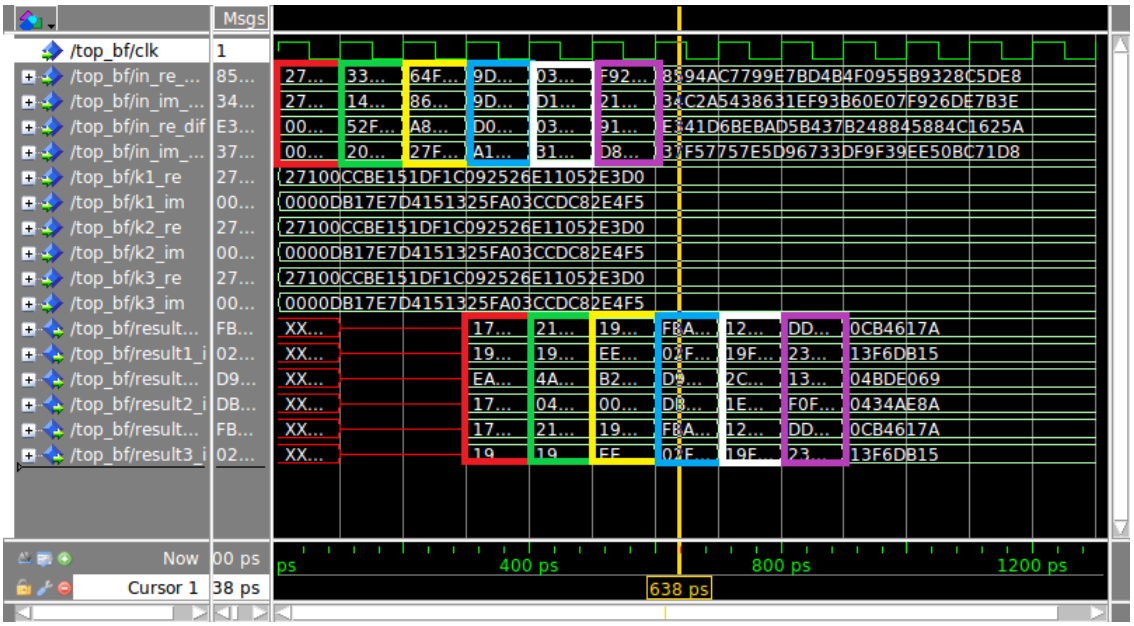
Po přijetí všech 8 paketů a předání dat do FIFO pamětí bude nastaven výstup ready což spustí beamformer. Vyčítání skončí po načtení posledních platných dat v paměti fifo. Z výstupní paměti budou procesorem vyčítány vypočtené paprsky. Před odesláním budou sestaveny do výstupního paketu obsahujícího původní hlavičku spolu s identifikací paprsku.

4 Ověření výsledků

4.1 RTL simulace

Pro prvotní ověření byla provedena simulace návrhu beamformeru. Celé řešení bohužel nelze simulovat z důvodu využití procesoru a dalších simulátorem nepodporovaných ip jader.

Pro porovnání byly využity komplexní váhy generované prostředím GNU Octave (MATLAB) a náhodně generovaná data v rámci rozsahu hodnot zpracovávaného datového typu *sint*32 (-32768 až +32767). S-parametry nebyly uvažovány. Data jsou po převodu do binárního formátu uložena do souboru ve formě skriptu pro simulátor modelsim. Ten po načtení skriptu nastaví hodnoty vstupů testované jednotky a provede krok testu. Vygenerovaná data jsou zároveň použita v další funkci pro výpočet výsledného paprsku jehož hodnota je v závěru porovnána s hodnotou výstupu simulátoru. Grafický příklad výstupu simulátoru je na obrázku4.1 Vstupy a výstupy označené stejnou barvou si odpovídají.



Obr. 4.1: Výstup simulace

Ze simulace je patrné, že modul je schopen pracovat se vstupními daty ve formě streamu a se zpožděním daným počtem registrů v datové cestě je schopen pracovat kontinuálně, což byl jeden z hlavních požadavků.

4.2 Reálná data

Pro simulaci reálného provozu byl tentokráte již celý návrh nahraný ve vývojovém kitu propojen pomocí sběrnice Ethernet s počítačem na kterém byl simulován reálný provoz radaru. S-parametry byly změřeny a jsou dodány firmou Retia a.s. Využitá data pochází z reálného chodu radaru a byla vytvořena záznamem datových toků programem wireshark. Při simulaci byl datový tok přehrán ze záznamu s využitím konzolové aplikace tcpreplay. Při testování aplikace byl obsah přijatých paketů vypisován na konzoli. Výpis přijatého paketu je na obrázku 4.2

```
Waiting for data...Received packet from 192.168.232.44:10562
odbeh: 11606671
rada:7
sloupec:0
souctoy kanal
dlouhy pulz
offset: 0
delka segmentu: 355
data:
00 00 ea ff 28 00 d7 ff f6 ff c5 ff ef ff ff ff fb ff 19 00 fc ff 24 00 f1 ff fe
ff f8 ff 02 00 05 00 08 00 f8 ff fd ff e6 ff f3 ff eb ff e7 ff 03 00 e0 ff e4 ff
e6 ff f0 ff 00 00 f1 ff 06 00 18 00 c3 ff f5 ff e5 ff e3 ff f7 ff dd ff f3 ff ba
ff d9 ff e4 ff c8 ff e9 ff 0a 00 e0 ff f9 ff f6 ff cd ff ee ff d2 ff d0 ff d5 ff
ba ff e3 ff d0 ff 10 00 e8 ff 1e 00 c5 ff f3 ff c0 ff df ff d6 ff 11 00 0d 00 0a
00 21 00 ec ff 18 00 ef ff d9 ff bf ff c7 ff d9 ff de ff 00 00 d7 ff 01 00 da ff
ff ff dc ff eb ff de ff e1 ff cf ff d1 ff b7 ff 8f ff f0 ff b7 ff e1 ff eb ff ff
ff ff ff 1d 00 e4 ff 26 00 de ff e5 ff eb ff b7 ff e4 ff c3 ff ec ff ea ff e2 ff
e8 ff e8 ff bf ff f8 ff e4 ff 10 00 d4 ff ed ff d0 ff e4 ff e0 ff ff ff de ff dd
ff f2 ff c1 ff f8 ff df ff c2 ff 03 00 ea ff 06 00 e7 ff ec ff dd ff ef ff 04 00
cc ff 18 00 c7 ff e2 ff dc ff b1 ff e7 ff bd ff da ff b3 ff 03 00 e4 ff 18 00 ec
ff e9 ff f0 ff fe ff de ff fe ff cb ff 12 00 be ff fb ff ab ff d2 ff e6 ff e5 ff
17 00 f4 ff 0b 00 03 00 e8 ff fb ff fb ff ca ff 13 00 f2 ff dc ff f6 ff 97 ff c3
ff c2 ff d5 ff e1 ff d2 ff f2 ff bd ff dc ff c6 ff cf ff c8 ff f1 ff dc ff ca ff
f6 ff d3 ff 06 00 d9 ff d8 ff e3 ff ce ff 02 00 f1 ff 11 00 0f 00 fe ff db ff d9
ff dc ff da ff 1b 00 ef ff d8 ff dc ff ba ff 08 00 e8 ff 05 00 d9 ff 17 00 b5 ff
e9 ff d2 ff d9 ff 21 00 f4 ff 1a 00 ec ff 0d 00 12 00 fe ff 01 00 dd ff fa ff da
ff 0c 00 e6 ff 0c 00 e6 ff f8 ff e5 ff df ff cc ff 22 00 02 00 0d 00 0f 00 e5 ff
06 00 f3 ff 08 00 12 00 1a 00 18 00 e5 ff
```

Obr. 4.2: Výpis přijatých udp paketů v konzole kitu

Jako problém se při běhu ukázalo být složení UDP paketů v záznamu. Ty mimo jiné obsahují informaci o fyzické adrese cíle, zjištěné před odesílám původního paketu. (obr.) , která je nadřazena ip adrese a kit tak nepřijímal žádná data. Řešením tohoto problému může být změna MAC adresy kitu na adresu původního signálového procesoru nebo jak tomu bylo i v tomto případě přepsání cílové adresy u všech paketů v záznamu na adresu kitu. Přepis byl proveden funkcí tcprewrite, která je součástí programu tcpreplay. Po přepsání bylo již možné pakety přijímat. Obrázek 4.2 zobrazuje výpis konzole vývojového kitu při odesílání reálného záznamu radaru.

Dalším problémem je přenos mezi HPS a FPGA, resp. ukládání přijatých dat do pamětí fifo. K tomuto účelu mají sloužit knihovny *altera_avalon_fifo.h, altera_avalon_fifo_util.h a altera_avalon_fifo_regs.h* dodávané firmou Intel. Tyto

knihovny jsou pravděpodobně součástí některého z vývojových prostředí firmy a nepodařilo se je získat. Jako náhrada knihoven byl pomocí simulátoru chipscope přes rozhraní JTAG testován alternativní způsob ukládání do pamětí zápisem do registrů mapovaných na vstup fifo pamětí. Do termínu odevzdání diplomové práce se bohužel nepodařilo vyřešit problém s přenosem dat. Komplexní ověření funkce na reálných datech tak nebylo provedeno.

5 Závěr

V rámci práce byla probrána teorie nezbytná k pochopení problematiky beamformingu a odvození rovnic pro výpočet tvarování svazků. V dalších kapitolách byl navržen beamformer na SoC Cyclone V. Příjem dat je realizován pomocí sběrnice ethernet v procesoru SoC. Pro korektní příjem UDP paketů kitem bylo nutné přepsat v paketech fyzickou adresu cíle z původního signálového procesoru. Synchronizaci datových streamů zajišťují paměti FIFO vybavené standardizovanou sběrnicí avalon streaming s potvrzením platnosti dat. Algoritmus tvarování byl navržen a ověřen RTL simulací. Implementovaný návrh je plně konfigurovatelný avšak nikoli za provozu, ale až po překladu. Teoretickou možností, jak zajistit konfigurovatelnost za provozu je využití maximálního počtu svazků pro daný obvod a čtení jen části výsledků. Obvod *5CSEMA4U23C6N* může počítat maximálně 3 svazky. Omezení vyplývá z počtu hardwarových násobiček. Při využití plánovaných obvodů rodiny Aria 10 bude možná realizace 6-70paprsků v závislosti na konkrétním obvodu. Výpadek některého z datových paketů způsobí ztrátu celého odběhu a ohlášní chyby. Ztráta odběhu neovlivní zpracování dalších dat. Ze zadání nebyl vzhledem k technickým problémům a vytížení kapacit firmy splněn bod týkající se odesílání dat. Jádro vytvořené na FPGA je díky využití standardizovaných rozhraní avalon možné využít v dalších aplikacích i v reálném provozu. Zbytek návrhu byl vytvořen pouze pro ověření funkce vzhledem k nedostupnosti vhodnějšího hardwaru. Pro cílovou aplikaci bude nutné nahrazení této části IP jádry pro příjem, zpracování a odesílání udp paketů pomocí sběrnice Ethernet požadovanou rychlostí.

Literatura

[1] Šebesta J.: *Radiolokace a radionavigace*. V Brně: Vysoké učení technické, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky, vyd. 1 vydání, 2004, ISBN 80-214-2482-6.

[2] Wolff, Ch.:*Radartutorial*,[Online],Lechfeld 1998[cit. 14.05.2019], URL:`http://www.radartutorial.eu/index.html#this`

[3] BEZOUŠEK, P. ŠEDIVÝ, P. *Radarová technika*. Praha: Vydavatelství ČVUT, 2004. ISBN 80-01-03036-9.

[4] Moernaut, G.J.K, Orban, D. *The Basics of Antenna Arrays*,Orban Microwave Products,[Online][cit.14.05.2019], URL:`https://orbanmicrowave.com/the-basics-of-antenna-arrays/`

[5] 5G Learning, *A Detailed Introduction to Beamforming*, [Online video], 6.5.2018,[cit.14.05.2019], URL:`https://www.youtube.com/watch?v=HKpQP8H4JRc`

[6] MAILLOUX, Robert J. *Phased array antenna handbook*. 2nd ed. Boston: Artech House, 2005. ISBN 978-1-58053-689-9.

[7] Kumar, G. *Antena arrays course*, [Online], IIT Bombai,[cit. 14.05.2019], URL:`https://nptel.ac.in/courses/108101092/Week-4-Antenna-Arrays-II.pdf`

[8] RUDGE, A. W. *The Handbook of antenna design*. [2nd ed.]. London, UK: P. Peregrinus on behalf of the Institution of Electrical Engineers, c1986. ISBN 0863410529.

[9] MAHAFZA, Bassem R. *Introduction to radar analysis*. Boca Raton: CRC Press, c1998. ISBN 0849318793.

[10] HANSEN, Robert C. *Phased array antennas*. New York: Wiley, c1998. ISBN 047153076X.

[11] SHERMAN, Samuel M. a David Knox BARTON. *Monopulse principles and techniques*. 2nd ed. Boston: Artech House, c2011. ISBN 9781608071746.

[12] LI, Jian a Petre STOICA. *Robust adaptive beamforming / edited by Jian Li and Petre Stoica*. Hoboken, NJ: John Wiley, 2006. ISBN 0471678503.

[13] Procházka, T., Bartoň, Z. *s-parametry*,[Online], Elektrorevue 29/2002, [cit.14.05.2019], URL:http://www.elektrorevue.cz/clanky/02029/index.html

[14] COOK, Charles E. a Marvin BERNFELD. *Radar signals: an introduction to theory and application*. Boston: Artech House, c1993. ISBN 0890067333.

[15] BALANIS, Constantine A. *Antenna theory: analysis and design*. 3rd ed. Hoboken: Wiley-Interscience, 2005. ISBN 978-0-471-66782-7.

[16] Shejbal, T.*Aktivní fázované anténní systémy pro přibližovací radary*, Pardubice 2016, Disertační práce, FEI UPCE, Školitel P.Bezoušek

[17] teasIC, *DE0-nano-SoC/Atlas-SoC Kit*, [Online katalogový list],2015, [cit.14.05.2019], URL:https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=163&No=941&PartNo=4

[18] Intel, *Cyclone V Device Overview*, [Online katalogový list],[cit.14.05.2019], URL:https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv_51001.pdf

[19] Intel, *Avalon Interface Specifications*,[Online katalogový list], [cit.14.05.2019], URL:https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf

[20] O'Donnell, R.M, *Radar system Engineering course*, [Online přednášky], IEEE New Hampshire 2010, [cit.14.05.2019], URL:http://aess.cs.unh.edu/Radar%202010%20PDFs/

[21] Mathworks: Čebyševovo okno.
URLhttps://www.mathworks.com/help/signal/ref/chebwin.html

[22] Mathworks: Taylor window function.
URLhttps://www.mathworks.com/help/signal/ref/taylorwin.html

[23] Pár, D.: Popis signálového zpracování Re3D, Retia a.s. - interní dokument.

[24] Strohmeier, M.; Smith, M.; Schäfer, M.; aj.: Assessing the Impact of Aviation Security on Cyber Power. 05 2016, 10.1109/CYCON.2016.7529437

[25] Vávra, V. L.: Multilaterační systémy.[Online], ERA a.s. 2017,[cit. 14.05.2019], URLhttps://slideplayer.cz/slide/13636938/

[26] Radar Handbook ; editor in Chief M. I. Skolnik. 2. ed. New York: McGraw-Hill Publishing Company, 1990.

[27] Intel, *On-Chip FIFO Memory Core*,[Online katalogový list],[cit 14.05.2019],URL:https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/qts_qii55002.pdf

[28] Intel, *Embedded Peripherals IP User Guide*,[Online katalogový list],[cit 14.05.2019],URL:https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_embedded_ip.pdf

[29] Mathworks, *Taylor window function*,[Online manuál],[cit 14.05.2019],URL:https://www.mathworks.com/help/signal/ref/taylorwin.htm

[30] Mathworks, *Chebyshev window function*,[Online manuál],[cit 14.05.2019],URL:https://www.mathworks.com/help/signal/ref/chebwin.htm

[31] ITU, *Nomenclature of the frequency and wavelengh bandsused in telecommunications*, [Online technický předpis],2015,[cit.14.05.2019],URL:https://www.itu.int/dms_pubrec/itu-r/rec/v/R-REC-V.431-8-201508-I!!PDF-E.pdf

[32] IEEE, *IEEE 521-2002 - IEEE Standard Letter Designations for Radar-Frequency Bands*,[Technický předpis],[cit.14.05.2019],URL:https://standards.ieee.org/standard/521-2002.html

Seznam symbolů, veličin a zkratek

FPGA	Field programmable gate array - Programovatelné hradlové pole
DSP	číslicové zpracování signálů – Digital Signal Processing
IEEE	Institut pro elektrotechnické a elektronické inženýrství – Institute of Electrical and Electronics Engineers
Soc	Systém na čipuů – System on Chip
HPS	Integrovaný procesor v systému – Hard Processor system
ARM	Procesor s jádrem firmy ARM – Advanced RISC Machine
GNU	Svobodný software inspirovaný unixem – GNU's Not Unix
RAM	Paměť s náhodným přístupem – Random acess memory
UDP	Internetový protokol bez záruky doručení – User Datagram Protocol
DTR	Modul pro digitalizaci signálu –
FIFO	Paměťová fronta – First in first out
IP	Licencované jádro/funkční blok – Intelectual Poperty
VHDL	Jazyk pro popis hadrware – VHSIC Hardware Description Language
HW	Pevně vytvořená část obvodu – Hardware
MAC	Fyzická adresa síťového zařízení – Media Access Control
A/D	Převodník spojitého signálu na číslicový – Analog Digital Converter
SP	Krátký pulz – Short pulse
LP	Dlouhý pulz – Long pulse
qsys	Software pro tvorbu top hierarchie projektu – Altera Platform designer

Seznam příloh

A Ručně psané zdrojové kódy

A.1 Beamformer

A.1.1 Balíček

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
package port_pkg is
  constant data_width : natural :=16;
  constant coef_exponent : natural :=4;
  constant data_len : natural :=8;
  subtype sint16 is std_logic_vector(data_width-1 downto 0);
  subtype sint32 is std_logic_vector ((2*data_width)-1 downto 0);
  subtype sint64 is std_logic_vector ((4*data_width)-1 downto 0);
  type sint16_arr is array(natural range <>) of sint16;
  type sint32_arr is array(natural range <>) of sint32;
  type sint64_arr is array(natural range <>) of sint64;
  function to_array16(input : std_logic_vector((data_width*data_len)-1
     downto 0))
  return sint16_arr;
end package port_pkg;

package body port_pkg is
  function to_array16(input : std_logic_vector((data_width*data_len)-1
     downto 0))
  return sint16_arr is
    variable tmp     :  sint16_arr(data_len-1 downto 0);
  begin
    for I in data_len downto 1 loop
    tmp(I-1):=input(((data_width*I)-1) downto (data_width*(I-1)));
    end loop;
    return tmp;
  end to_array16;
end package body port_pkg;
```

A.1.2 Komplexní násobička

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use  work.port_pkg.all;

entity cplx_mult is
```

```
port
  (
    clk : in std_logic;
    in_re     : in std_logic_vector ((data_width-1) downto 0);
    in_im     : in std_logic_vector ((data_width-1) downto 0);
    k_re     : in std_logic_vector ((data_width-1) downto 0);
    k_im     : in std_logic_vector ((data_width-1) downto 0);
    result_r : out std_logic_vector (((2*data_width)-1) downto 0);
    result_i : out std_logic_vector (((2*data_width)-1) downto 0)
  );
end entity;

architecture rtl of cplx_mult is
  signal tmp_rr : signed (((2*data_width)-1) downto 0);
  signal tmp_ri : signed (((2*data_width)-1) downto 0);
  signal tmp_ir : signed (((2*data_width)-1) downto 0);
  signal tmp_ii : signed (((2*data_width)-1) downto 0);
  signal tmp_r : std_logic_vector (((2*data_width)-1) downto 0);
  signal tmp_i : std_logic_vector (((2*data_width)-1) downto 0);
begin
  process (in_re, in_im,tmp_rr,tmp_ri,tmp_ir,tmp_ii,k_re,k_im)
  begin
    tmp_rr <=(others=>'0');
    tmp_ri <=(others=>'0');
    tmp_ir <=(others=>'0');
    tmp_ii <=(others=>'0');
    tmp_rr <= signed(in_re)*signed(k_re);
    tmp_ri <= signed(in_re)*signed(k_im);
    tmp_ir <= signed(in_im)*signed(k_re);
    tmp_ii <= signed(in_im)*signed(k_im);
    tmp_r <= std_logic_vector(tmp_rr-tmp_ii);
    tmp_i <= std_logic_vector(tmp_ri+tmp_ir);
  end process;

  process(clk)
  begin
    if rising_edge(clk) then
    result_r<=tmp_r;
    result_i<=tmp_i;
    end if;
  end process;

end rtl;
```

A.1.3 Komplexní sčítačka

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use  work.port_pkg.all;

entity cplx_adder_n is
  port
    (
      clk : in std_logic;
      r_in     : in sint32_arr(data_len- 1 downto 0);
      im_in    : in sint32_arr(data_len- 1 downto 0);
      result_r : out sint32;
      result_i : out sint32
    );
end entity;

architecture rtl of cplx_adder_n is
begin

  process (clk,r_in, im_in)
    variable tmp_rv : sint32 :=(others=>'0');
    variable tmp_iv : sint32 :=(others=>'0');
  begin
      tmp_rv :=(others=>'0');
      tmp_iv :=(others=>'0');
    for i in 0 to data_len-1  loop
      tmp_rv := std_logic_vector(signed(r_in(i)) + signed(tmp_rv));

      tmp_iv := std_logic_vector(signed(im_in(i)) + signed(tmp_iv));
    end loop;
      result_r <= tmp_rv;
      result_i <= tmp_iv;
  end process;
end rtl;
```

A.1.4 bf_unit (výpočet 1 kanálu)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use  work.port_pkg.all;

entity bf_unit is
  port
  (
    clk : in std_logic;
```

```
    in_re     : in std_logic_vector(((data_len*data_width)-1) downto 0);
    in_im     : in std_logic_vector(((data_len*data_width)-1) downto 0);
    k_re      : in std_logic_vector(((data_len*data_width)-1) downto 0);
    k_im      : in std_logic_vector(((data_len*data_width)-1) downto 0);
    result_r : out sint32;
    result_i : out sint32
  );
end entity;

architecture rtl of bf_unit is
  component cplx_mult is
    port
    (
      clk : in std_logic;
      in_re     : in std_logic_vector ((data_width-1) downto 0);
      in_im     : in std_logic_vector ((data_width-1) downto 0);
      k_re     : in std_logic_vector ((data_width-1) downto 0);
      k_im     : in std_logic_vector ((data_width-1) downto 0);
      result_r : out std_logic_vector (((2*data_width)-1) downto 0);
      result_i : out std_logic_vector (((2*data_width)-1) downto 0)
    );
  end component;

  component cplx_adder_n is

    port
    (
      clk : in std_logic;
      r_in      : in sint32_arr(data_len- 1 downto 0);
      im_in     : in sint32_arr(data_len- 1 downto 0);
      result_r : out sint32;
      result_i : out sint32
    );

  end component;

  signal    in_re_s    :  sint16_arr(data_len-1 downto 0);
  signal    in_im_s    :  sint16_arr(data_len-1 downto 0);
  signal    k_re_s     :  sint16_arr(data_len-1 downto 0);
  signal    k_im_s     :  sint16_arr(data_len-1 downto 0);
  signal postmul_r : sint32_arr (data_len-1 downto 0);
  signal postmul_i : sint32_arr (data_len- 1 downto 0);
  signal div_r : sint32;
  signal div_i : sint32;
begin

  in_re_s<=to_array16(in_re);
```

```
  in_im_s<=to_array16(in_im);
  k_re_s<=to_array16(k_re);
  k_im_s<=to_array16(k_im);


  GEN_MULT:
    for I in data_len-1 downto 0 generate
      mx:cplx_mult
      port map(clk => clk,
      in_re=>in_re_s(I),
      in_im=>in_im_s(I),
      k_re=>k_re_s(I),
      k_im=>k_im_s(I),
      result_r=>postmul_r(I),
      result_i=> postmul_i(I)
      );
    end generate GEN_MULT;

  add:cplx_adder_n
    port map( clk=>clk,
          r_in=>postmul_r,
          im_in=>postmul_i,
          result_r=>div_r,
          result_i=>div_i
    );
  result_r<=div_r;
  result_i<=div_i;
end rtl;
```

A.1.5 beamformer

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use  work.port_pkg.all;

entity top_bf is
  port
  (
    clk : in std_logic;
    in_re_sum     : in std_logic_vector(((data_len*data_width)-1) downto
        0);
    in_im_sum     : in std_logic_vector(((data_len*data_width)-1) downto
        0);
    in_re_dif     : in std_logic_vector(((data_len*data_width)-1) downto
        0);
```

```
    in_im_dif     : in std_logic_vector(((data_len*data_width)-1) downto
        0);
    k1_re     : in std_logic_vector(((data_len*data_width)-1) downto 0);
    k1_im     : in std_logic_vector(((data_len*data_width)-1) downto 0);
    k2_re     : in std_logic_vector(((data_len*data_width)-1) downto 0);
    k2_im     : in std_logic_vector(((data_len*data_width)-1) downto 0);
    k3_re     : in std_logic_vector(((data_len*data_width)-1) downto 0);
    k3_im     : in std_logic_vector(((data_len*data_width)-1) downto 0);
    result1_r : out std_logic_vector ((2*data_width)-1 downto 0);
    result1_i : out std_logic_vector ((2*data_width)-1 downto 0);
    result2_r : out std_logic_vector ((2*data_width)-1 downto 0);
    result2_i : out std_logic_vector ((2*data_width)-1 downto 0);
    result3_r : out std_logic_vector ((2*data_width)-1 downto 0);
    result3_i : out std_logic_vector ((2*data_width)-1 downto 0)
  );
end entity;

architecture rtl of top_bf is
  signal in_re_sum_tmp     :  std_logic_vector(((data_len*data_width)
     -1) downto 0);
  signal in_im_sum_tmp     :  std_logic_vector(((data_len*data_width)
     -1) downto 0);
  signal in_re_dif_tmp     :  std_logic_vector(((data_len*data_width)
     -1) downto 0);
  signal in_im_dif_tmp     :  std_logic_vector(((data_len*data_width)
     -1) downto 0);
  signal k1_re_tmp     :  std_logic_vector(((data_len*data_width)-1)
     downto 0);
  signal k1_im_tmp     :  std_logic_vector(((data_len*data_width)-1)
     downto 0);
  signal k2_re_tmp     :  std_logic_vector(((data_len*data_width)-1)
     downto 0);
  signal k2_im_tmp     :  std_logic_vector(((data_len*data_width)-1)
     downto 0);
  signal k3_re_tmp     :  std_logic_vector(((data_len*data_width)-1)
     downto 0);
  signal k3_im_tmp     :  std_logic_vector(((data_len*data_width)-1)
     downto 0);

  signal result1_r_tmp :  sint32;
  signal result1_i_tmp :  sint32;
  signal result2_r_tmp :  sint32;
  signal result2_i_tmp :  sint32;
  signal result3_r_tmp :  sint32;
  signal result3_i_tmp :  sint32;

  component bf_unit is
```

```
    port
    (
      clk : in std_logic;
      in_re     : in std_logic_vector(((data_len*data_width)-1) downto
          0);
      in_im     : in std_logic_vector(((data_len*data_width)-1) downto
          0);
      k_re      : in std_logic_vector(((data_len*data_width)-1) downto
          0);
      k_im      : in std_logic_vector(((data_len*data_width)-1) downto
          0);
      result_r : out sint32;
      result_i : out sint32
    );
  end component;

begin

  process(clk,in_re_sum,in_im_sum,in_re_dif,in_im_dif,k1_re,k1_im,k2_re
      ,k2_im,k3_re,k3_im)
  begin
    if rising_edge(clk) then
      in_re_sum_tmp<=in_re_sum;
      in_im_sum_tmp<=in_im_sum;
      in_re_dif_tmp<=in_re_dif;
      in_im_dif_tmp<=in_im_dif;
      k1_re_tmp<=k1_re;
      k1_im_tmp<=k1_im;
      k2_re_tmp<=k2_re;
      k2_im_tmp<=k2_im;
      k3_re_tmp<=k3_re;
      k3_im_tmp<=k3_im;
      result1_r<=result1_r_tmp;
      result1_i<=result1_i_tmp;
      result2_r<=result2_r_tmp;
      result2_i<=result2_i_tmp;
      result3_r<=result3_r_tmp;
      result3_i<=result3_i_tmp;
    end if;
  end process;
  bff1:bf_unit
    port map(clk =>clk,
      in_re=>in_re_sum_tmp,
      in_im=>in_im_sum_tmp,
      k_re=>k1_re_tmp,
      k_im=>k1_im_tmp,
      result_r=>result1_r_tmp,
```

```
      result_i=>result1_i_tmp
    );
  bff2:bf_unit
    port map(clk =>clk,
      in_re=>in_re_dif_tmp,
      in_im=>in_im_dif_tmp,
      k_re=>k2_re_tmp,
      k_im=>k2_im_tmp,
      result_r=>result2_r_tmp,
      result_i=>result2_i_tmp
    );
  bff3:bf_unit
    port map(clk =>clk,
      in_re=>in_re_sum_tmp,
      in_im=>in_im_sum_tmp,
      k_re=>k3_re_tmp,
      k_im=>k3_im_tmp,
      result_r=>result3_r_tmp,
      result_i=>result3_i_tmp
    );
end rtl;
```

A.2 Příjem UDP

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<arpa/inet.h>
#include<sys/socket.h>
#define BUFLEN 512
#define PORT 10562

int main(void)
{
  struct sockaddr_in si_me, si_other;
  socklen_t s, slen = sizeof(si_other) , recv_len;
  char buf[BUFLEN];

  s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP))

  memset((char *) &si_me, 0, sizeof(si_me));

  si_me.sin_family = AF_INET;
  si_me.sin_port = htons(PORT);
  si_me.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
    if( bind(s , (struct sockaddr*)&si_me, sizeof(si_me) ) == -1)
    {
      die("bind");
    }

    while(1)
    {
      memset(buf,'\0', BUFLEN);
      printf("Waiting for data...");
      fflush(stdout);

    recv_len = recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *) &si_other
        , &slen))

      printf("Received packet from %s:%d\n", inet_ntoa(si_other.sin_addr)
          , ntohs(si_other.sin_port));
      printf("odbeh: %d\n",buf[0]<<24|buf[1]<<16|buf[2]<<8|buf[3]);
      printf("rada:%d\n",buf[4]);
      printf("sloupec:%d\n",buf[5]);
      if (buf[6]==1){
        printf("souctoy kanal\n");
      }else{
        printf("roydilovy kanal\n");
      }
      if (buf[7]==1){
        printf("kratky pulz\n");
      }else{
        printf("dlouhy pulz\n");
      }
      printf("offset: %d\n",buf[8]<<8|buf[9]);
      printf("delka segmentu: %d\n",buf[10]<<8|buf[11]);
      printf("data:\n");
      for(int i=12;i<recv_len;i++){
        printf("%02x ",(unsigned char)buf[i]);
      }
      printf("\n");
    }
    return 0;
}
```

A.3 MATLAB

A.3.1 Komplexní váhy

```
## Lv -- antena element distance (default 73.1mm)
##
## ap -- aperture angle (default 15 deg)
##
## f -- signal frequency (default 3120MHz)
##
## N -- number of array elements (default 8)
##
## mod -- transmitter signal modulation [array of N] elements (
    default chebwin(8,25))
##
## sv -- array of beam angles
## @end deftypefn
## Author: kuba <kuba@kuba-HP-ProBook-4530s>
## Created: 2019-03-19

function [an,bn] = bfk (Lv=73.1e-3, ap=15,f=3120e6,N=8,mod=chebwin
    (8,25),beam=0)

  if isempty(Lv)
    Lv = 73.1e-3;
  end
  if isempty(ap)
    ap = 15;
  end
  if isempty(f)
    f = 3120e6;
  end
  if isempty(N)
    N = 8;
  end
  if isempty(mod)
    mod = chebwin(8,25);
  end
  if isempty(beam)
    beam = 0;
  end

  for cnt=1:length(beam)
    an(1:N,cnt)=exp(2*i*pi*(0:N-1)*((Lv*f)/299792458)*sin((beam(cnt)-ap
        )*(pi/180)));

    cn=2*transpose(((2*(1:N))-N-1)/(N-1));
```

```
    bn(1:N,cnt)=cn(1:N).*an(1:N,cnt);
  end
endfunction
```

A.3.2 Beamforming

```
function [B_sum,B_az,B_ele] = beamform (an, bn, sn_sum,sn_az,sn_ele,
    values_sum, values_dif)
  for cnt=1:size(values_sum,2)
    A_sum =(an./sn_sum).*values_sum;
    A_az =(an./sn_az).*values_dif;
    A_ele =(bn./sn_ele).*values_sum;
  end

  B_sum = sum (A_sum);
  B_az = sum (A_az);
  B_ele = sum (A_ele);
endfunction
```

B Obsah přiloženého CD

Součástí práce je CD s archivem celého projektu i textu práce.

- `latex` ..... text práce
  - `loga` ..... loga školy a fakulty
    - `FEKT-spec-color.eps`
    - `FEKT-spec-color.pdf`
    - `logolink-op_vavpi.png`
    - `RE-spec-color.eps`
    - `RE-spec-color.pdf`
    - `SIX_logo_zahlavi.png`
  - `obrazky` ..... ostatní obrázky
  - `pdf` ..... pdf stránky generované informačním systémem
    - `student-desky.pdf`
    - `student-titulka.pdf`
    - `student-zadani.pdf`
  - `text` ..... zdrojové textové soubory
    - `literatura.tex`
    - `prilohy.tex`
    - `reseni.tex`
    - `uvod.tex`
    - `vysledky.tex`
    - `zaver.tex`
    - `zkratky.tex`
  - `navod-sablona_FEKT.pdf` ..... návod na používání šablony
  - `obhajoba.tex` ..... hlavní soubor pro sazbu prezentace k obhajobě
  - `readme.txt` ..... soubor s popisem obsahu CD
  - `sablona.tex` ..... hlavní soubor pro sazbu kvalifikační práce
  - `thesis.sty` ..... balíček pro sazbu kvalifikačních prací

- `prace` — složka projektu
  - `vhdl` — soubory pro FPGA
    - `port_pkg.vhd`
    - `cplx_mult.vhd`
    - `bf_unit.vhd`
    - `top_bf.vhd`
    - `KOEF_RAM_regs_pkg.vhd`
    - `KOEF_RAM_regs.vhd`
    - `qsys_top.vhd`
  - `c` — soubory pro HPS
    - `server.cpp`
    - `cegrd.c`
    - `regwr.c`
  - `ATLAS_SOC_GHRD.rbf` — Konfigurační soubor FPGA
  - `beamformer.qpf` — hlavní soubor projektu quartus
  - `beamformer.qsys` — hlavní soubor projektu Platform designer
  - `DP_Barta.tar` — Arciv celého projektu včetně generovaných souborů